An algebraic framework for reasoning about privacy

by

Solofomampionona Fortunat Rajaona

Dissertation approved for the degree of Doctor of Philosophy in the Faculty of Science at Stellenbosch University

Department of Mathematics,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Promoter: Prof. J.W. Sanders

March 2016

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

S. F. Rajaona March 2019

Abstract

An algebraic framework for reasoning about privacy

S. F. Rajaona
Department of Mathematics,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Dissertation: PhD (Maths)

March 2016

In this thesis, we study a formal programming language and algebraic techniques to analyse computational systems that considers data confidentiality and hidden computations. The reasoning techniques are based on the refinement of programs (Back and von Wright, Carroll Morgan). The underlying logic is a first-order $S5_n$ epistemic logic that distinguish between objects and concepts – of the family of Melvin Fitting's First Order Intensional Logic. We give a relational semantics and a weakest-precondition semantics to prove the soundness of programming laws. The laws for confidentiality refinement extends those of Carroll Morgan's Shadow Knows refinement calculus, whereas the laws for reasoning about knowledge derives mostly from the Public Announcement Logic. As applications for knowledge dynamics, we study the classical puzzles of the Three Wise Men and the Muddy Children by means of the programming laws; and as an application for reasoning about confidentiality and anonymity, we give a sketch of formal analysis of the Anonymous Cocaine Auction Protocol.

Acknowledgements

I would like to express my sincere gratitude to my doctoral advisor Jeff Sanders for his guidance, support, and friendship. I thank the German Academic Exchange Service (DAAD), and the African Institute for Mathematical Sciences (AIMS) - South Africa for their financial support.

I thank all the friends who helped my family have a memorable stay in Cape Town. Particularly, the Ralaivaosaona, the Rabenatoandro, the FTMCTM members, Gillian Hawkes, the Quakers, Liane Greeff and Roy McGregor, and all AIMS students and staff.

Misaotra ny fianakaviana sy ny havana rehetra any Madagasikara izay nitrotro am-bavaka sy nankahery hatrany anay telo mianaka. Ary ny fitiavanao ry Sophie, ny fahendrenao ry Mahefa, sy ny faharetanareo no nahavitako asa tsara sy nahatonga ahy ho tompon'andraikitra.

Dedication

To Sophie and Mahefa,

Contents

Chapter 1

Introduction

1.1 Overview

This thesis focuses on reasoning formally about what agents in a group can learn about the program resources from an execution of a program. Here a *program* does not necessarily mean a computer program. A program may refer to any system of interest that we can describe using a syntax close to that of a programming language.

Example 1.1 *The systems that we call programs might be computer programs, communication protocols, card games etc.. As such, a resource of the program might be a public or a private key, a password, a set of cards, confidential data, etc.. The agents might be the users of a program, the players of a game, wise men, muddy children, cocaine dealers etc..*

We are interested in what each individual in a group of agents can learn from the execution of a program. But that does not mean that the programs we are interested in are meant only to give information to the agents. A program usually has other purposes that are not related to the knowledge of the agents. We refer to these as *functional requirements*. And we refer to the requirements related to the flow of information as *confidentiality requirements*.

Example 1.2 *A functional requirement in a voting or an anonymous auction protocol is election of a winner whilst a confidentiality requirement is that no individual vote or bid is revealed during the process.*

The functional requirements of a program are related to the factual changes whereas the confidentiality requirements to be related to information flow. We define the factual changes to be the changes made by the program on its resources. Information flow is the change in the knowledge of an agent that can partially observe the program and its resources.

Example 1.3 *A communication protocol. Changing a public key is a factual change. Announcing the value of a public key is a change of information.*

Remark 1.1 A statement of knowledge might be recorded as a program variable but this is not practical. For example, let a propositional variable k_a be true if agent a knows the color of its hat v_a. On one hand, whenever there is an operation leaking v_a to agent a the variable k_a needs to be reassigned (simultaneously). On the other hand, reassigning k_a, e.g., $k_a := 1$, can correspond to any operation that reveals v_a (e.g., announcing publicly v_a or reassigning v_a).

In this dissertation, we provide a common framework to reason about the factual changes induced by a program and the flow of information to each individual in a group of agents. Our aim is to use only algebraic laws of program refinement, similar to those given for standard programs (see e.g., [27, 20, 21]).

In the following sections, we give an overview of different formalisms constituting, influencing, or closely related to this work. In the final section of this introduction, we provide a description of this thesis highlighting its scope and its contributions.

1.2 Refinement of programs

Practically, program refinement is the transformation of what is to be achieved into how to achieve it. There might be more than one way to achieve a specification. We say that a specification is nondeterministic. Refining a specification makes it more deterministic, thus closer to a language instructable to a computer.

In a formal program refinement framework [2, 27], specifications are understood to be also programs, though not always executable. In that setting *refinement* is a relation between specifications. A program P is said to be refined by Q, denoted $P \sqsubseteq Q$, when Q preserves all the logically expressible properties of P.

$$P \sqsubseteq Q \;\equiv\; \text{if } P \text{ guarantees } \varphi \text{ then } Q \text{ guarantees } \varphi \tag{1.2.1}$$

or reading the contraposition

$$P \sqsubseteq Q \;\equiv\; \text{if } Q \text{ can breach } \varphi \text{ then } P \text{ can breach } \varphi$$

The logical property φ might be for example a first-order logic formula such as $\exists\, n \in \mathbb{N} \bullet v = 2^n$ for some program variable v. This notion of program refinement has been used for reasoning about sequential programs, concurrency, and probabilistic systems. Morgan [28] defined an ignorance-preserving refinement to reason about confidentiality. Originally, refinement was designed for sequential programs and the concern was functional correctness. The following describes program semantics, which are mathematical frameworks that are used to formalise (1.2.1).

Semantics give mathematical meaning to programs. Relational semantics associates a program P to a relation between an initial state s_i and any final state s_f that can be obtained after executing P from s_i. It is equivalent to take the meaning of P to be a function $[\![P]\!]$ taking an initial state s_i to the set of possible final states that can be reached from s_i (denotational semantics, see e.g., [23]). For standard programs, i.e., without issue of confidentiality, the values of the program variables determine a state and refinement is defined by

$$P \sqsubseteq Q \;\equiv\; [\![P]\!].\sigma \supseteq [\![Q]\!].\sigma \quad \text{for any initial state } \sigma \tag{1.2.2}$$

In predicate transformer semantics, a command is interpreted as a function between set of states (or predicates), e.g., Dijkstra's weakest precondition [10]. Given a predicate φ and a command P,

$$wp.P.\varphi \qquad\qquad (\textit{Dijkstra's weakest precondition})$$

is the weakest predicate that an initial state needs to satisfy so that by executing P from it the resulting state satisfies φ.

Below is the meaning of refinement in the weakest precondition semantics.

$$P \sqsubseteq Q \;\equiv\; wp.P.\varphi \Rightarrow wp.Q.\varphi \quad \text{for any predicate } \varphi \tag{1.2.3}$$

The refinement calculus framework [27, 2] enables formal program verification and construction. By means of programming laws, a specification is refined step by step, to give an implementation. Specifications and program codes are all part of a unique space of "programs". The programming space involves operators such as nondeterministic choice $\sqcap$ and sequential composition $\fatsemi$, and it is ordered by the refinement relation $\sqsubseteq$. The program models (predicate transformer or denotational models) are used only to prove the soundness of the laws used in the refinement.

1.3 Refinement and confidentiality

The refinement calculus for classical programs is not suitable for direct use in reasoning about confidentiality (see e.g., [28]). Some valid classical refinements can breach confidentiality requirements as in the *Refinement Paradox* (Example 2.2). Morgan proposed, in his *Shadow Knows* refinement framework [28], an ignorance-preserving refinement that avoids the paradox and keeps most of the classical program algebra. His work was continued in [23, 25, 24] and was used to formally derive security protocols.

The characterisation of ignorance-preserving refinement extends that of classical programs in (1.2.1). An agent (or attacker) a is assumed, and refinement has to take into account what can be learnt by a.

$$P \sqsubseteq Q \quad \equiv \quad \left(\text{and} \quad \begin{array}{l} \text{if } P \text{ guarantees } \varphi \text{ then } Q \text{ guarantees } \varphi \\ \text{if } P \text{ guarantees to hide } \varphi \text{ from } a \text{ then } Q \text{ do so} \end{array}\right) \tag{1.3.1}$$

or reading the contraposition

$$P \sqsubseteq Q \quad \equiv \quad \left(\text{and} \quad \begin{array}{l} \text{if } Q \text{ can breach } \varphi \text{ then } P \text{ can breach } \varphi \\ \text{if } Q \text{ can allow } a \text{ to learn } \varphi \text{ then } P \text{ can do so.} \end{array}\right) \tag{1.3.2}$$

1.4 Knowledge in modal logic

As in the work of Morgan for ignorance-preserving refinement, we will reason about knowledge using the modal operator K. This particular logic is called *epistemic logic* [19] and is part of the big family of modal logic [7]. Epistemic logic, like any other modal logic, is a system built on top of another logical language to describe the modal operator. It provides a proof system based on the choice of axioms and inference rules. It also provides a relational model that interprets the logical formulas. In this section, we discuss about propositional modal logic, which is the most dominant in literature[1].

Axioms

To define an epistemic logic on top of propositional logic, axioms from the following set

$$\mathrm{K}(\phi \to \psi) \to (\mathrm{K}\phi \to \mathrm{K}\psi) \qquad \text{(Modal distribution)}$$
$$\mathrm{K}\phi \to \phi \qquad \text{(Axiom of necessitation)}$$
$$\mathrm{K}\phi \to \mathrm{K}\phi \qquad \text{(Positive introspection)}$$
$$\neg\mathrm{K}\phi \to \mathrm{K}\neg\mathrm{K}\phi \qquad \text{(Negative introspection)}$$

and the inference rule

$$\mathrm{K}(\phi \to \psi), \mathrm{K}(\phi) \vdash \mathrm{K}\psi \tag{1.4.1}$$

are added to the axioms and inference rules of propositional logic. Taking all four axioms above defines the epistemic logic S5. The system S5 assumes the knowledge an ideal agent, or an agent with perfect knowledge.

Possible worlds models

Another characteristic of the modal logic family is the use of a possible worlds model called Kripke structure [19] or variants of it. In propositional modal logic, for a given set P of propositions, a Kripke structure or model is a triple (W, R, V) where

- W is a set of possible worlds,
- each R is a relation (accessibility relation) between worlds of W,
- V is a valuation function from W to P that determines which propositions are true at each world.

[1]See e.g., [5] for that observation.

Logical semantics interprets the validity of logical formulas on this type of model.

Following the Correspondence Theorem (see e.g., [5]), the choice of taking epistemic logic S5 corresponds to considering Kripke models whose accessibility relations are equivalence relations.

Multiple agents and Common Knowledge

Epistemic logic can also be used to study the knowledge of a group of agents about some facts (basic information) and their knowledge about the knowledge of other agents (higher-order information). Thus we assume a group $\mathcal{A}$ agents. And for each agent $a \in \mathcal{A}$, we define a modal operator K_a. For example, the formula $\mathrm{K}_a(v = 0)$ reads "agent a knows that $v = 0$". For a multi-agent epistemic logic, the axioms are generalised for the family of modalities $(\mathrm{K}_a)_{a \in \mathcal{A}}$ And in the model, a family of accessibility relations $(R_a)_{a \in \mathcal{A}}$ is considered.

An important notion in the study of multi-agent knowledge is common knowledge, which is illustrated in the following example from [4]

Example 1.4 (On Common Knowledge) *I (JFAK van Benthem) approach you in a busy Roman street, A.D. 180, intent on rescuing my former general Maximus, now held as a gladiator, and ask:*

> *Q: Is this the road to the Colosseum?*

As a well-informed and helpful Roman citizen, you answer truly:

> *A: Yes.*

... By asking the question, I convey to you that I do not know the answer, and also, that I think it possible that you do. This information flows before you have said anything at all. Then, by answering, you do not just convey the topographical fact to me. You also bring it about that you know that I know, I know that you know I know, etc.. This knowledge up to every finite depth of mutual reflection is called common knowledge. It involves a mixture of factual information and iterated information about what others know ....

Given a set of agents $A \subseteq \mathcal{A}$, the formula $\mathrm{C}_A\phi$ reads "it is common knowledge to agents in A that ϕ". Whereas the formula $\mathrm{E}_A\phi$ reads "every agent in

A knows that ϕ", and is equivalent to $\bigwedge\{a : A \bullet \mathrm{K}_a\phi\}$. We note that the truth of $\mathrm{C}_A\phi$ implies the truth of $\mathrm{E}_A\mathrm{E}_A \ldots \mathrm{E}_A\phi$ for any number of repetitions, and hence the truth of $\mathrm{K}_{a_0}\mathrm{K}_{a_1} \ldots \mathrm{K}_{a_{n-1}}\phi$ for any n and any choice of the a_i's. The semantic interpretations and axioms for common knowledge can be found for example in [12].

1.5 Logic of knowledge and information change

Epistemic models describe the state of knowledge of agents. Different approaches exist to reason about how the knowledge of agents evolve. Two particular branches of study exists: *Dynamic Epistemic Logic* (DEL, [3, 17, 11]) studies mechanisms that update epistemic models and Interpreted Systems/Epistemic Temporal Logic (ETL, [12, 32]) incorporates an epistemic structure to linear or branching time models; see [6] for a comparative study and the merging of the two frameworks.

In this dissertation, we make use of only a subset of DEL namely the *Public Announcement Logic* (PAL, see [33, 18]). In PAL, a formula $[\phi]\psi$ ($\langle\phi\rangle\psi$) [2] means after every (some) announcement of ϕ, holds ψ. The announcements considered in PAL are public and truthful (only true formulas are announced).

Most existing tools that extend epistemic logic to more complex systems make use of an underlying propositional logic. All the tools for reasoning about knowledge we presented so far uses propositional logic. Yet, in the reasoning about program and information flow state properties are better represented and sometimes require quantification.

1.6 Logic of knowledge in first-order

In order to reason about program refinement and knowledge, we would like to have a theory of program refinement where the properties of a state are expressed in some epistemic logic formula. It would be ideal, to follow the classical approach of program refinement [27, 2] and to have a predicate logic rather than propositional as a basis of the programming techniques. Moreover,

[2] [33] uses the notation $\phi + \psi$ for $\langle\phi\rangle\psi$

such a feature would give us more expressive power. For example, in using an operator KV for knowing the value of a variable.

> A richer epistemic predicate logic can make further important distinctions about knowledge of properties of individual objects, such as $\exists x K_a \varphi(x)$ ("*de re* knowledge") versus $K_a \exists x \varphi(x)$ ("*de dicto* knowledge"). J. van Benthem in *Modal Logic for Open Minds* [5].

But first-order modal logic is not as popular as its propositional counterpart and there is no single direction for the researches made on it. Several problems arise both in the proof systems and in the models when adding quantification and variables to modal logic (see e.g., [15, 5]), e.g., the distinction between *de dicto* and *de re* knowledge.

De dicto/De re

EXAMPLE 1.5 *Consider two program variables $h : hid.a$ and $v : vis.a$ where $a \in \mathcal{A}$. In some state of the program where $v = 0$ and a knows that $v = h$, we expect to make the following deduction.*

$$\left(\wedge \begin{array}{l} \mathrm{K}_a(h = v) \\ \mathrm{K}_a(v = 0) \end{array} \right) \Rightarrow \mathrm{K}_a(h = 0). \tag{1.6.1}$$

Although this deduction is correct, it does not come from simple equal to equal substitution. For if we can just make equal to equal substitution then: suppose we are at another state where $h = 19$, we have the following.

$$\left(\wedge \begin{array}{l} \mathrm{K}_a(h = h) \\ h = 19 \end{array} \right) \Rightarrow \mathrm{K}_a(h = 19) \tag{1.6.2}$$

The deduction in 1.6.2 means that agent a knows h also at that state even if a is not supposed to be given any hint about h.

In fact the false deduction in 1.6.2 comes from the fact that knowing $v = h$ does not have the same meaning as knowing $h = h$. The latter is a knowledge *de dicto* and the former a knowledge *de re*. This distinction is extensively explained in philosophy with examples like "the morning star and the evening star" or "Hesperus and Phosphorus" or "Superman and Clark Kent"; see e.g.,

[15] which gives also a historical context on how philosophers have approached this problem.

We need a first-order logical system that allows formulas that have the same meaning as (1.6.1) and disallows formulas that have the same meaning as (1.6.2). This motivates our use of Fitting's First-Order Intensional Logic (FOIL) [16, 13, 15]. FOIL distinguishes between object and concept, and makes use of predicate abstraction to distinguish the *de dicto* and the *de re* reading of modal formulas; see Chapter 3.

1.7 Description of this thesis

Scope

We focus on what a program can achieve and what the agents can learn about the variables from the execution. Thus, in our programs, we do not make explicit who is performing an action in the program. We assume that the environment performs the program and the agents are observers. In a system in which the identity of the agent who perfoms an action carries extra information, that information is made implicit in the program itself. For example, a program "Alice sends a message to Bob" is captured by "The environment sends a message of Alice to Bob" or "A message of Alice was sent to Bob". See more examples in the next chapter.

We assume the environment carrying the actions does not discuss issues of interference on a shared resource. But that assumption allows us to reason about anonymity without difficulty, e.g., "a message is sent to Bob". This feature is used in Chapter 6 to formally describe an anonymous auction protocol. Efficiency of a program is outside of our formalism. Quantitative information analyses, like probabilities are not part of it but we believe they can be added later. That is also the case for recursive and iterative program constructs.

Contributions

This work is an extension of the works of Morgan [28] in the use of algebraic techniques to reason about privacy in sequential programs. Although Morgan and McIver could develop and analyse multi-agents systems such as security protocols [23, 25, 26], Morgan's Shadow model of computation is based on a single agent point of view and is not suitable to interpret elaborate epistemic logical formulas such as the nesting of modalities. Our main contribution is to combine their programming language and techniques with a full (multi-modal) epistemic logic. The resulting ignorance-preserving refinement of programs can be used for both secure protocol analysis and dynamic epistemic logic reasoning.

We added more commands to Morgan's programming language and allowed epistemic logical expressions. The latter required us to define an appropriate first-order logic of knowledge. Our logic is a First Order Intensional Logic (FOIL, [13]) enriched with a visibility type on the flexible intensions (concepts). In our setting of ideal agents (assuming the axioms of S5), the logic was given a multi-dimensional logic model (in the sense of [22]). Using this logic, we propose a first-order version of the Public Announcement Logic ([11]).

We developed models of computations appropriate for multi-modal epistemic logical formulas: the first interprets a program as a relation between epistemic models and the second is a predicate transformer semantics (Dijkstra's weakest precondition) in which the underlying logic is a multi-modal epistemic logic. In particular, we use the Public Announcement Logic as a basis for our weakest precondition semantics.

We made possible the use of Morgan and McIver's techniques to study scenarios involving nested knowledge, e.g., I know that you know that .... The classical puzzles of the Three Wise Men and the Muddy Children [12] were analysed using program algebraic laws. We also give a formal description of the Cocaine Auction Protocol [35] that demonstrates the efficacy of the programming language to reason about anonymity and privacy in communication protocols.

Plan

Chapter 2 gives the assumptions and the syntax of the programs with examples on how to use them to model certain kinds of system. Chapter 3 defines the first-order modal logic used throughout the thesis. Chapter 4 presents the two programming models and their connection. Chapter 5 gives a collection of programming laws. Chapter 6 presents the case studies: the Three Wise Men, the Muddy Children puzzles, and the Cocaine Auction Protocol. Chapter 7 concludes the work.

Chapter 2

Program algebra

In this chapter, we explain the assumptions and define the language adopted in this thesis. We give examples using the language to model systems of information flow and confidentiality-aware computations.

2.1 Assumptions

A program variable is either hidden or visible to each agent.

This assumption distinguishes programming with privacy from standard imperative programs. A variable must be specified with the set of agents that can see it or alternatively with the set of agents that cannot see it. Thus a declaration of a variable in the program must have one of the forms

$$\mathsf{vis}_a\, v \;\text{ or }\; \mathsf{hid}_a\, v. \tag{2.1.1}$$

Inside the scope of $\mathsf{vis}_a\, v$, an agent a can observe the value stored in v during the execution of the program; i.e., a can observe the changes of the variables visible to it after each atomic step of the program. In contrast, inside a scope of $\mathsf{hid}_a\, v$, agent a never sees the content of v. Yet it is possible that a infers the value of v because of its knowledge of the code; that is our following assumption.

The program code is common knowledge to all agents.

Knowledge of the program code means that the agents know the code of the program to be executed. This assumption was shown by Morgan to be useful for reasoning about information flow using program refinement. It is also a required assumption when we model the program states to form an epistemic model: if every state in the model satisfies a logical formula ϕ, then ϕ is a theorem, thus every agent knows it, by the axiom of necessitation (see Section 1.4). Therefore, for our reasoning of multi-agents systems, we need to assume that the program is a theorem– a common knowledge to the agents.

We distinguish between non-atomic and atomic programs.

Morgan [28] introduced atomic execution to limit the power of the agents. An agent can observe the values of the variables visible to him only between atomic steps of a program and not within them. To assume that some steps are hidden to an agent, we need to make the program atomic. In our setting of multi-agent systems, we define a notion of atomicity for each agent. A program inside atomic brackets $\langle\!\langle\rangle\!\rangle_a$ is understood as atomic to agent a.

Nondeterministic and atomic choice. In standard programs the nondeterministic choice $P \sqcap Q$ can execute either be P or Q. Morgan introduced confidentiality requirement into program refinement. He showed in [28]that, if the nondeterministic choice $\sqcap$ has to satisfy structural algebraic laws, then it has to be interpreted as an explicit choice. In contrast, an atomic choice needs to be defined for modelling a hidden choice. This can achieved, for example, with $\langle\!\langle P \sqcap Q\rangle\!\rangle_a$ or with $:\in$ as we will see.

Example 2.1 *Modelling a vote. The choice by a voter should ideally be atomic (visible only to him). His choice whether to vote or not is explicit.*

The distinction between an explicit and an atomic nondeterministic choice provides a program algebra that makes the *refinement paradox* invalid.

Example 2.2 (The refinement paradox) *the classical setting, i.e., without confidentiality requiremens, we have the following refinements.*

$$v := -1 \sqcap v := 1 \quad \sqsubseteq \quad v := 1$$

and

$$v := -1 \sqcap v := 1 \quad \sqsubseteq \quad v := -1$$

Both $v := 1$ and $v := -1$ are possible implementations of the nondeterministic choice in the left hand side. Such refinements are forbidden if we require that v is hidden to some agent a. But in fact, these should be forbidden only if the choice between the two implementations is hidden from a. Thus, in our setting,

$$v := -1 \sqcap v := 1 \quad \sqsubseteq \quad v := 1$$

and

$$v := -1 \sqcap v := 1 \quad \sqsubseteq \quad v := -1$$

but

$$v :\in \{-1, 1\} = \langle\!\langle v := -1 \sqcap v := 1 \rangle\!\rangle \quad \not\sqsubseteq \quad v := 1$$

and

$$v :\in \{-1, 1\} = \langle\!\langle v := -1 \sqcap v := 1 \rangle\!\rangle \quad \not\sqsubseteq \quad v := -1.$$

2.2 Program syntax

In the following we will define the language of programs, namely the syntax for expressions and the syntax of commands. We refer to Chapter 3 for the definition of the more elaborate logical expressions.

2.2.1 Vocabulary and expressions

To define our programming language we fix the following sets of symbols

- $\mathcal{A}$ of all agents denoted by $a, b, \ldots$; subsets of agents are denoted by $A, B, \ldots$
- $\mathcal{C}$ of constants denoted by $c, d, \ldots$
- $\mathcal{R}$ of relations denoted by $R, S, \ldots$
- $\mathcal{F}$ of functions denoted by $f, g, \ldots$

In addition we have program variable symbols, which we denote by $v, w, \ldots$. The set of program variables may vary. In some context we specify the programs to have a common set $\mathcal{P}$ of global variables.

$P ::=$	skip	(No operation)
$\mid$	$v := e$	(Assignment)
$\mid$	$v :\in E$	(Invisible choice)
$\mid$	$P \fatsemi P$	(Sequential composition)
$\mid$	$P \lhd \phi \rhd P$	(Conditional)
$\mid$	$P \sqcap P$	(Visible choice)
$\mid$	$\mathsf{ann!}\,\phi$	(Announce pubicly that p)
$\mid$	$\mathsf{rev}_A\ e$	(Reveal e to agents in a)
$\mid$	$[\![\, \mathsf{vis}_a\ v \bullet P]\!]$	(Local variable)
$\mid$	$\langle\!\langle P \rangle\!\rangle_a$	(Atomic execution)
$\mid$	$\mathsf{assert}\,\phi$	(Assertion)
$\mid$	abort	(Divergence)
$\mid$	magic	(Miracle)

Table 2.1: Syntax of programs

Definition 2.1 An *expression* is defined in BNF form as follows

$e ::=$	c	(Constant)
$\mid$	v	(Program variable)
$\mid$	$f(e_0, e_1, \ldots e_N)$	(Function)

Definition 2.2 A *logical expression* is a closed formula as defined in Section 3.1.1. The constant concepts symbols in $\mathcal{I}$ are the constants in $\mathcal{C}$ and the program variables symbols.

2.2.2 Visible and hidden variables declaration

The command $[\![\, \mathsf{vis}_A\ v \bullet P]\!]$ introduces a new variable v that is visible only to the agents in A. The new variable can be used only within its scope, which determined by the brackets $[\![\cdot]\!]$. We adopt the following notations.

$$\begin{aligned} \mathsf{vis}_a\, v &= \mathsf{vis}_{\{a\}}\, v \\ \mathsf{vis}\, v &= \mathsf{vis}_{\mathcal{A}}\, v = \mathsf{hid}_{\{\}}\, v \\ \mathsf{hid}\, v &= \mathsf{hid}_{\mathcal{A}}\, v = \mathsf{vis}_{\{\}}\, v \end{aligned}$$

2.2.3 Atomic choice, assignment

An atomic choice $v :\in E$ nondeterministically changes the value of v to the value of an expression from a non-empty set E of expressions. This type of command is assumed to always terminate and to be executed immediately (atomically). Thus, only the agents that can see the variable v know its value after the choice, unless E is a singleton.

The assignment command $v := e$ is a particular case of the previous atomic choice command, when the set E is a singleton.

$$v := e \;=\; v :\in \{e\} \tag{2.2.1}$$

2.2.4 Skip, magic, and abort

The command skip performs nothing. It terminates immediately and changes no variable. The command abort is a computation that may fail to terminate or may terminate in any arbitrary final state. In our setting, abort is the most insecure command, as it can possibly produce a state where a secret is leaked. The command magic is a computation that is never enabled.

$$\mathsf{skip} \;=\; v := v \tag{2.2.2}$$

2.2.5 Explicit atomicity

For a program P and a set A of agents, the command $\langle\!\langle P \rangle\!\rangle_A$ "makes P atomic except for the group A". It ensures that agents outside A cannot observe any change of the program variables occurring during the execution of P. These agents outside A see the changes induced by P as a single step from the initial values – before P– to the final values – after P.

We adopt the following notations.

$$\begin{aligned} «P»_a &= «P»_{\{a\}} \\ «P» &= «P»_{\mathcal{A}} \\ «P»_{\{\}} &= P \end{aligned}$$

We note that whilst Morgan and McIver [29, 23] allows the atomic brackets to apply only to classical programs, we give it a more expressive power. For instance, we allow the nesting of atomic brackets. We emphasize that the brackets $«»_A$ reads as "prevent agents outside A to see all the intermediate values inside the brackets" rather than "allow agents in A to see all intermediate values inside the brackets". For example, it is possible that inside the brackets $«»_a$ (which executes atomically only for agents other than a), some intermediate values of $v : vis.a$ might be hidden to a (possible occurence of $«»_{\overline{a}}$ inside $«»_a$), see the following example.

Example 2.3 *Consider two variables x, y visible to all and a variable s hidden to all. In the program*

$$\langle\!\langle y :\in \{0,1\} \fatsemi \langle\!\langle x := s \fatsemi x := x \oplus y \rangle\!\rangle_A \fatsemi y := 0 \rangle\!\rangle_{\bar{A}},$$

agents in A cannot learn s. The outside atomic brackets prevent them to see the intermediate value of x that leaks the value of s. But agents outside A can learn s even if they also cannot see the intermediate value of x that leaks s (the inner brackets prevent them). Indeed, because y is visible to agents outside A, and they are not prevented to see the intermediate value of y (after the first $\fatsemi$), they can infer the value of s. They do so by looking at the final value of x and the intermediate value y (we have $x = s \oplus y$, thus $s = x \oplus y$ after the inner atomic block).

2.2.6 Public announcement command

If the logical expression ϕ is true, the command $\mathsf{ann!}\,\phi$ "reveals that ϕ" or "announces that ϕ" to all agents. It changes no program variable. If ϕ is false, the command is not enabled, thus ϕ cannot be announced. We have the following law resulting from the assumption that the program code is common knowledge.

Law 2.1 $\mathsf{ann!}\,\phi \;=\; \mathsf{skip} \lhd \phi \rhd \mathsf{magic}$

A simultaneous announcement of finitely many formulas is possible using their conjunction. Because of the assumption on the common knowledge of the program, if a program reveals that ϕ to an agent, then it is common knowledge that it reveals that ϕ. Therefore we do not indicate to which agent an announcement is made. To model a private announcement, we need to use the command ann! with other commands; see Subsection 2.3.1.

Law 2.2 $\langle\!\langle \mathsf{ann!}\,\phi \rangle\!\rangle_A \;=\; \mathsf{ann!}\,\phi$

Remark 2.1 We note that with this definition of ann! , announcing a false formula cannot be enabled: it is miraculous (see e.g., [27] for miraculous programs). For standard programs ann! is equivalent to coercion. It is also possible to define an announcement command that is equivalent to assertion in standard commands i.e.,

$$\mathsf{skip} \lhd \phi \rhd \mathsf{abort}\,. \tag{2.2.3}$$

In this case, the program aborts when false is announced. But the use of coercion is more appropriate for us. For instance, in expressing the revelation made in executing a conditional expression; see Law 2.5.

2.2.7 Revelation of an expression

Given a group A of agents and an expression e, the command $\mathsf{rev}_A\{e\}$ "reveals e to A". It changes no (global) variables. A publication is an atomic command but a revelation is not. That is because what an agent learn from an atomic program is from its observation of the global variables before and after the atomic program, and from its knowledge of the program code. The following laws concerns the atomic execution of rev .

Law 2.3 $\langle\!\langle \mathsf{rev}_A\{e\} \rangle\!\rangle_B \;=\; \mathsf{rev}_{A\cap B}\{e\}$

Law 2.4 $\mathsf{rev}_{\{\}}\{e\} \;=\; \mathsf{skip}$

2.2.8 Sequential composition

The command $P \fatsemi Q$ executes P and if P terminates, it executes Q.

2.2.9 Nondeterministic choice, conditional choice

The command $P \lhd \phi \rhd Q$ reads as "P if ϕ else Q". It executes P if ϕ is true and Q if ϕ is false. The following law results from the assumption that the program code is common knowledge.

Law 2.5 $P \lhd \phi \rhd Q \;=\; \left(\sqcap \begin{array}{l} \mathsf{ann!}\,\phi \fatsemi P \\ \mathsf{ann!}\,\neg\phi \fatsemi Q \end{array} \right)$

2.2.10 Remarks on the syntax

Assertion

In the assertion $\langle\phi\rangle$, ϕ is a logical expression of program variables. If ϕ is true, the program continues with the next command, else the program diverges. As for coercions we have

$$\mathsf{assert}\,\phi \;=\; \text{«}\,\mathsf{skip} \lhd \phi \rhd \mathsf{abort}\,\text{»}. \tag{2.2.4}$$

Specifications

We could have included a specification command $[\hat{\psi}]$[1] in the style of [21, 27, 2] in this programming language. In a specification $[\hat{\psi}]$, the logical expression $\hat{\psi}$ involves the program variables and their dashed versions (we put the hat to indicate that it involves dashed variables). For example, in the scope of a program variable v, the command $[\hat{\psi}.v.v']$ specifies that if there are values of v' that make $\hat{\psi}$ true, then the program is enabled, changing nondeterministically v to one of the values v', and leaving any other variable unchanged. If there is no possible value, the program is not enabled.

An atomic specification «$[\hat{\psi}]$» has the same effect on the program variables as $[\hat{\psi}]$ but executes atomically and specifies, in particular, that there is no other leak of information apart from that specified in the formula $\hat{\psi}$.

Example 2.4 *In the scope of two global variables u, v and an agent a, non-atomic specification allows insecure refinement such as*

$$[v' \in \{0,1\}] \;\sqsubseteq\; (v := 0 \sqcap v := 1).$$

[1]Not to confuse the square brackets with a public announcement formula $[\phi]\psi$, see Section 3.3

However, the atomic version allows only refinement by another atomic program, ensuring no leak of extra information. We have

$$«[v' \in \{0,1\}]» \;=\; v :\in \{0,1\}.$$

When relating nondeterminism to atomic and a non-atomic specification, we have

$$\begin{aligned} [\hat{p} \vee \hat{q}] \;&=\; [\hat{p}] \sqcap [\hat{q}] \\ «[\hat{p} \vee \hat{q}]» \;&\not\sqsubseteq\; «[\hat{p}]» \sqcap «[\hat{q}]» \end{aligned}$$

An example of this is

$$[x' = 0] \sqcap [x' = 1] \not\sqsubseteq [x' = 0 \;\vee\; x' = 1]$$

which is the analogue expressed in specifications of

$$x := 0 \sqcap x := 1 \not\sqsubseteq x :\in \{0,1\}$$

When the logical expression inside a specification does not involve dashed variables we have a coercion. In our setting atomic coercion is synonymous with the ann! command

$$«[\psi]» \;=\; «\,\mathsf{skip} \triangleleft \psi \triangleright \mathsf{magic}\,» \;=\; \mathsf{ann!}\,\psi \tag{2.2.5}$$

Recursion and Iteration

These commands are yet to be included in our language. However, we note that programs with loops were considered in Morgan and McIver's ignorance-preserving refinement [26].

2.3 Modelling with programs

2.3.1 Modelling hidden computation

How do we model a situation where we want the program code to be hidden from some agents? We have to use a metaprogram in which, the program code is a choice among others that the metaprogram can takei, and that choice is hidden to the agent in question. An example to hide a program P from an agent a is

$$\llangle P \sqcap \mathsf{skip} \rrangle_a . \tag{2.3.1}$$

If such program is executed, although the agent a knows the code, it does not know whether P was executed or not. But one might ask *if a knows the code (2.3.1) wouldn't a be suspicious that P might have run?* This is a property of an ideal agent: it cannot ignore its ignorance.

2.3.2 Signed revelations

We can construct commands that substitute a publication or revelation made by a specific agent. Such individual publication or revelation gives the extra information that the agent knows the value of the expression it is communicating or that it knows the formula it is revealing. We have

$$b\,\mathsf{rev}_a\{e\} \;\widehat{=}\; \mathsf{ann!}\{\mathrm{KV}_b\, e\} \mathbin{;} \mathsf{rev}_a\{e\} \tag{2.3.2}$$
$$b\,\mathsf{ann!}\,p \;\widehat{=}\; \mathsf{ann!}\{(p \wedge \mathrm{K}_b p)\}. \tag{2.3.3}$$

We note that from this definition, if in a program P, an agent reveals something it does not know, then P is not enabled, P is magic.

2.3.3 Modelling anonymity

As we have seen above, the commands for revelations do not make any assumption on who is making the revelation. We make use of these to reason about anonymous broadcast in our case study of the Cocaine Auction Protocol of Stajano and Anderson [35]; see Chapter 6. Using anonymous communication commands such as $\mathsf{ann!}[]$ and rev should facilitate the study of cryptographic protocols:

> ... new ideas come from challenging fossilised axioms: it is needlessly limiting to design all cryptographic protocols under the unexamined assumption that communications must perforce be point-to-point ... Stajano and Anderson in [35]

Summary

The syntax of basic programs presented in this chapter appeared in the setting of imperative programs (e.g., in [2] and [27]). The distinction between visible and hidden variables the `rev` command, the explicit atomicity command are taken from [28]. Some of the syntax was already used in [26] and [23]. But instead of using first-order predicates for logical expressions, we use a first order logic of knowledge that is the object of following chapter.

Chapter 3

Logics

This chapter presents the syntax of the logic used in this dissertation. In particular the logical expressions in the programming syntax of the previous chapter makes use of this logic. Throughout this dissertation, we will assume a first-order epistemic logic that describe the knowledge of ideal agents. Thus we assume a logic obeying the system of axioms S5 (see Section 1.4). This logic will be given a concrete Kripke model, which is a special case of Kripke model for S5 logics. Our study of these models will allow us to study relations between models, notably the refinement between models, the modal equivalence between models, and the updates of a model. The chapter will end with our proposed first order version of the Public Announcement Logic.

3.1 First-order epistemic logic

The First Order Intensional Logic (FOIL) is a family of logic introduced by Fitting in [16, 13, 15]. FOIL logics distinguish between concepts and objects to overcome the *de dicto/de re* problem (see Section 1.6). Concepts were defined to be functions from possible worlds to the object domain. Different types of the logic exist according to taking these functions to be total or partial, and to taking the object domain to vary or to be constant across the possible worlds. We will now explore the definitions and terminology of FOIL.

The following example is taken straight from [16].

Example 3.1 (On designation and existence) *Consider the concept "The King of France in 1700". Such a concept designates an object, which is Louis XIV.*

But the object Louis XIV does not exist now in 2016. Thus in the world of 2016, the King of France in 1700 designates a non-existing object. Consider the concept "The present King of France". Such a concept does not designate at all now in 2015.

Fitting (see e.g., [15]) used this example to explain the difference between designation and existence in FOIL. Designation is a property of terms whilst existence is a property of objects. In our reasoning of programs, we do not expect to assume non-existing objects. The possible worlds are possible runs of the same program. And it is reasonable to assume that the domain of a program does not vary for different executions.

Example 3.2 (On designation and existence) *In a game of Texas Hold Em Poker, the object domain is the set of 52 cards. It should be the case that this object domain stays the same in all possible scenarios of the game. Let the concepts (u_a, v_a) be the cards held by player a. In a point of the game when the dealer is still shuffling the cards, it does not make sense to talk about (u_a, v_a), i.e., these concepts do not designate. The concepts (u_a, v_a) will designate only when a takes his cards from the table. In a reasonable poker game, the two cards are always among the 52. The object domain is always the deck of 52 Poker cards. We would not think about a situation where the player a is holding two Tarot cards instead.*

But different runs of a program may introduce different local variables. In reasoning about confidentiality, these variables cannot be discarded directly. Local variables can leak information to the agents. To treat local variables, we have to allow concepts to be partial functions. These observations lead us to consider a FOIL logic that assumes:

- concepts to be partial functions: terms might not designate
- a constant object domain: terms always designate an existing object.

The meaning, in terms of programs, of "a term designate at a world", is that a term can be evaluated at the state in question. Thus,

> the requirement "a term always designates an existing object" corresponds to the programming requirement "a declared program

variable must have been initialised".

Now we introduce the syntax of the FOIL logic that we will use.

3.1.1 Syntax

Our vocabulary consists of:

- Concept or intension variable symbols, denoted by x, y, z, ...
- Object variable symbols, denoted by X, Y, Z, ...
- Constant concept symbols, denoted $n, m, \ldots$ when naming a rigid concept, and v, u, w, ... when naming a flexible concept
- Constant objects symbols, denoted by $N, M, \ldots$
- Function symbols, denoted by f, g,...
- Relation symbols, denoted by R, S,...
- Agent symbols, denoted by a, b,... (A, B,... denote sets of agents.)

Definition 3.1 (Term) A term is defined in BNF form as follows

$$
\begin{aligned}
t ::= \; & X && \text{(Object variable)}\\
\mid \; & x && \text{(Concept variable)}\\
\mid \; & c && \text{(Constant concept)}\\
\mid \; & f(t_0, t_1, \ldots, t_{N-1}) && \text{(Function)}
\end{aligned}
$$

Objects are of type O, and concepts are of type I. If a function f is of type $(\tau, \ldots, \tau)$ for $\tau \in \{O, I\}$ then the term $f(t_0, \ldots, t_{N-1})$ is of type τ. This means that we distinguish between object terms and concept terms.

Definition 3.2 (Atomic formula) An atomic formula has the form $R(t_0, t_1, \ldots, t_{N-1})$ where R is a relation of type $(\tau, \ldots, \tau)$, where $\tau \in \{O, I\}$.

Definition 3.3 (Formula) A formula is defined in BNF form as follows

$$\begin{array}{rll}
\phi ::= & R(t_0, t_1, \ldots, t_{N-1}) & \text{(Atomic formula)} \\
& \mid \neg\phi & \text{(Negation)} \\
& \mid \phi \circ \phi & \text{(Binary composition)} \\
& \mid \forall x \bullet \phi & \text{(Universal quantification on concepts)} \\
& \mid \exists x \bullet \phi & \text{(Existential quantification on concepts)} \\
& \mid \mathrm{K}_a\, \phi & \text{(K-modal)} \\
& \mid (\lambda X \bullet \Phi).t & \text{(Predicate abstract)}
\end{array}$$

Symbol ϕ designate any formula and Φ designate a formula with objects only.

We do not consider symbols for constant objects because we can refer to them by using the concept associated. For example, the equality $X = 2$ between two objects can be substituted by $(\lambda Y \bullet X = Y).2$. In the latter, 2 is the (rigid) concept that is interpreted as the object 2 in every world. The formula $(\lambda Y \bullet X = Y).2$ reads as the object X is equal to the object designated by the concept 2 (which is also the object 2). This means that we have a rigid concept associated to each object of the domain.

Each N-place function is either a function on concepts or a function on objects. To distinguish between the two kinds, functions are given a type in addition to their arity. A N-place function on concepts has a type that is the $N+1$-uple $(I, \cdots, I)$. And a N-place function on objects has a type that is the $N+1$-uple $(O, \cdots, O)$. The same applies for relations: the type of a N-place relation is either the N-uple $(I, \cdots, I)$ or the N-uple $(O, \cdots, O)$.

It is possible to allow functions and relations to have arguments of mixed objects and concepts. In that case, their type are tuple of Os and Is (see e.g., [15]). For simplicity, we replace such functions and relations by their equivalents but with only concepts arguments. This is possible since we assumed to have a rigid concept associated to each object. For example, consider the addition $N \hat{+} c$ between an object variable N and a concept variable c. The addition $N \hat{+} c$ can be replaced by the addition $n \dot{+} c$ between two concepts, where n is a rigid concept variable associated to the object variable N. Using predicate abstraction, $N \hat{+} c$ is equivalent to $(\lambda X \bullet N + X).c$ whereas $n \dot{+} c$ is equivalent to $(\lambda X, Y \bullet Y + X).(n, c)$. For instance, in a possible world where the concept c designates the object 2, the concept $n\dot{+}c$ designates the object $N+2$.

Definition 3.4 (Vocabulary) A vocabulary $\mathcal{V}$ consists of a set $\mathcal{I}$ of constant concepts symbols, a set $\mathcal{F}$ of function symbols, and a set $\mathcal{R}$ of relation symbols.

Definition 3.5 (Formula, sentence) A $\mathcal{V}$-formula is a formula in which every constant symbols is in the vocabulary $\mathcal{V}$. A $\mathcal{V}$-sentence is a $\mathcal{V}$-formula with no free variable.

3.1.2 Models

Definition 3.6 (Standard model) A standard model on a vocabulary $\mathcal{V}$ is a tuple

$$\mathfrak{M} = \langle \mathsf{S}, (\sim_a)_{a\in\mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}[\![\,.\,]\!]\rangle$$

where

1. S, the set of possible worlds, is a given set
2. $(\sim_a)_{a\in\mathcal{A}}$, the accessibility relations on S, are equivalence relations on S: for each a, $\sim_a \subseteq \mathsf{S}\times\mathsf{S}$
3. D_o, the domain of objects, is a given set
4. D_c, the domain of concepts, is a subset of partial functions in $\mathsf{S} \nrightarrow \mathsf{D}_o$
5. $\mathfrak{M}[\![\,.\,]\!]$, the interpretation function, interprets the symbols in $\mathcal{V}$:

   – for each $c \in \mathcal{I}$: $\mathfrak{M}[\![c]\!] \in \mathsf{D}_c (\subseteq \mathsf{S} \nrightarrow \mathsf{D}_o)$
   – for each N-place function $f \in \mathcal{F}$ of type $(\tau,\ldots,\tau)$, where $\tau \in \{O, I\}$: $\mathfrak{M}[\![f]\!] \in \mathsf{S} \nrightarrow \mathsf{D}_\tau^N \to \mathsf{D}_\tau$
   – for each N-place relation $R \in \mathcal{R}$ of type $(\tau,\ldots,\tau)$, where $\tau \in \{O, I\}$: $\mathfrak{M}[\![R]\!] \in \mathsf{S} \nrightarrow \mathsf{D}_\tau^N$

The set $\mathcal{I}$ of concepts is partitioned into the subset $\mathcal{I}_f$ of flexible concepts and the subset $\mathcal{I}_r$ for rigid concepts. The same partition applies for $\mathcal{F}$ and $\mathcal{R}$.

The previous definition of a FOIL model is similar to most first-order modal logic Kripke models. The definition of the starts by giving a frame $\langle \mathsf{S}, (\sim_a$

$)_{a\in\mathcal{A}}\rangle$. The frame consists of a particular nonempty set S of possible worlds and a family of relations given on S. Then, domains D_o and D_c are associated with the frame. And a function $\mathfrak{M}[\![\,.\,]\!]$ is defined to interpret the symbols in the vocabulary, at each world of S.

In this type of model, that we call standard model, to each possible world is associated an interpretation of the constant symbols. But there might be two different worlds having the same interpretation of all the constant symbols. We define concrete models to be models that have a correspondence between possible worlds and the interpretation of the constant symbols. This means that, in a concrete model, a possible world is uniquely defined by its interpretation of the constant symbols.

Definition 3.7 (Attributes) The world attributes in the vocabulary $\mathcal{V}$ of a model $\mathfrak{M}$ are the symbols in a subset $\mathsf{A} \subseteq \mathcal{V}$, such that, the interpretation of the symbols in A completely determines a world in the model $\mathfrak{M}$.

Because the interpretation of certain symbols in $\mathcal{V}$ does not vary from world to world – in this case we say that the interpretation is rigid –, a world is completely determined by its interpretation of the non-rigid symbols. Thus, the attributes correspond exactly to the symbols that interpret non-rigid (or flexible) constants, functions, and relations.

In our study, the only flexible constant symbols that we consider are the flexible concepts symbols. Functions and relations symbols are all assumed to be interpreted rigidly. Thus, we consider models over a vocabulary $\mathcal{V} = \mathcal{I}_f \cup \mathcal{I}_r \cup \mathcal{F} \cup \mathcal{R}$ (we omit the subscript r for $\mathcal{F}$ and $\mathcal{R}$). If $\mathfrak{M} = \langle \mathsf{S}, (\sim_a)_{a\in\mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}[\![\,.\,]\!]\rangle$ is such a model, then S is determined by the interpretation of flexible symbols. Thus S is determined by the restriction of the interpretation function $\mathfrak{M}[\![\,.\,]\!]$ to the flexible symbols $\mathcal{I}$, denoted $\mathcal{I}_f \lhd \mathfrak{M}[\![\,.\,]\!]$. An element of S is of the form $\mathsf{s} : \mathcal{I}_f \to \mathsf{D}_o$. We will see that each accessibility relation $\sim_a$ is also determined by the interpretation of a particular predicate $vis.a$ on $\mathcal{I}_f$.

Definition 3.8 (Concrete model) Consider a vocabulary $\mathcal{V} = (\mathcal{I}_r \cup \mathcal{I}_f, \mathcal{F}, \mathcal{R})$ that contains a predicate $vis.a$ for every $a \in \mathcal{A}$. A concrete model on $\mathcal{V}$ is a tuple $\mathfrak{M} = \langle \mathsf{S}, (\sim_a)_{a\in\mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}[\![\,.\,]\!]\rangle$ where

1. D_o, the domain of objects, is a given set
2. S, the possible worlds, is a subset of partial functions in $\mathcal{I}_f \nrightarrow \mathsf{D}_o$
3. D_c, the domain of concepts, is a subset of partial functions in $\mathsf{S} \nrightarrow \mathsf{D}_o$
4. $\mathfrak{M}[\![\,.\,]\!]$, the interpretation function, interprets the symbols in $\mathcal{V}$:
   - for each $v \in \mathcal{I}_f$: $\mathfrak{M}[\![v]\!] \in \mathsf{D}_c$ and $\mathfrak{M}[\![v]\!].\mathsf{s} = \mathsf{s}.v$ for $\mathsf{s} \in \operatorname{dom} v$
   - for each $c \in \mathcal{I}_r$: $\mathfrak{M}[\![c]\!] \in \mathsf{D}_c$ and is total and constant
   - for each N-place function $f \in \mathcal{F}$ of type $(\tau, \ldots, \tau)$, where $\tau \in \{O, I\}$: $\mathfrak{M}[\![f]\!] \in \mathsf{D}_\tau^N \to \mathsf{D}_\tau$
   - for each N-place relation $R \in \mathcal{R}$ of type $(\tau, \ldots, \tau)$, where $\tau \in \{O, I\}$: $\mathfrak{M}[\![R]\!] \in \mathsf{D}_\tau^N$
5. $(\sim_a)_{a \in \mathcal{A}}$, the accessibility relations on S, are determined by the visibility of the elements of $\mathcal{I}_f$, i.e., by $\mathcal{I}_f \lhd \mathfrak{M}[\![vis.a]\!]$.

We will give the definition of an accessibility relation $\sim_a$ only after defining term evaluation and designation in a model. The definition of term evaluation and designation do not require the accessibility relations.

3.1.3 Semantics

Remark 3.1 (Notation) We denote elements in the semantics by *sans serif* characters. Possible worlds in S are denoted by *sans serif* characters $\mathsf{s}, \mathsf{t}, \ldots$ Elements of the domains D_c are denoted by *sans serif* characters $\mathsf{m}, \mathsf{n}, \ldots$. Elements of the domains D_o are denoted by *sans serif* characters $\mathsf{M}, \mathsf{N}, \ldots$. Functions on the domains are denoted by $\mathsf{f}, \mathsf{g}, \ldots$. Usual mathematical operators will be understood from the context.

Definition 3.9 (Assignment) An assignment is a mapping that assigns to each object variable an element of D_o, and assigns to each concept variable an element of D_c. We denote assignments by Greek letters $\mu, \nu, \ldots$.

Definition 3.10 (Evaluation) Consider $\mathfrak{M}$ to be a model and μ to be an assignment. The evaluation of a term t is the partial function $\mathfrak{M}_\mu[\![t]\!]$ from S

to D_o satisfying

$$\mathfrak{M}_\mu[\![X]\!] = \lambda \mathsf{s} \bullet (\mu.X)$$

$$\mathfrak{M}_\mu[\![x]\!] = \mu.x$$

$$\mathfrak{M}_\mu[\![v]\!] = \mathfrak{M}[\![v]\!]$$

$$\mathfrak{M}_\mu[\![f.(t_0, \ldots, t_{N-1})]\!].\mathsf{s} = \mathfrak{M}[\![f]\!].(\mathfrak{M}, \mu[\![t_0]\!], \ldots, \mathfrak{M}_\mu[\![t_{N-1}]\!])$$

where X is an object variable, x is a concept variable, v is a constant concept, and f is a function.

We note that the evaluation of an object is a total function. An object variable is evaluated to a unique element of D_o in all possible worlds. However, the evaluation of a concept is a partial function. The evaluation of the free variables is just the assignment μ.

Definition 3.11 (Designation) Given a model $\mathfrak{M}$, an assignment μ of the free variables, and a state s, we say that t *designates* at s when $\mathsf{s} \in \operatorname{dom} \mathfrak{M}_\mu[\![t]\!]$ and the object designated by t is $\mathfrak{M}_\mu[\![t]\!].\mathsf{s}$. We denote $D.t$ the predicate that is true at just the states where t designates.

Definition 3.12 (Accessibility relation) In a $\mathcal{V}$-model $\mathfrak{M} = \langle \mathsf{S}, (\sim_a)_{a \in \mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}[\![\,.\,]\!] \rangle$, we define for $\mathsf{s}, \mathsf{t} \in \mathsf{S}$, and for each agent a, the accessiblity relation $\sim_a$ by

$\mathsf{s} \sim_a \mathsf{t}$ if for all v in $\mathcal{I}_f$ such that $v \in \mathfrak{M}[\![vis.a]\!]$:

v designates at s iff v designates at t and $\mathfrak{M}[\![v]\!].\mathsf{s} = \mathfrak{M}[\![v]\!].\mathsf{t}$.

In the following, a triple $(\mathfrak{M}, \mathsf{s}, \mu)$ consists of a model $\mathfrak{M}$, a possible world s of $\mathfrak{M}$, and an assignment μ of the free variables.

Definition 3.13 (Truth of a formula) We define the truth of a formula at a triple $(\mathfrak{M}, \mathsf{s}, \mu)$ as follows

1. $\mathfrak{M}, \mathsf{s}, \mu \vDash R.(t_0, \ldots, t_{N-1})$ iff all t_i designates at s for μ and $(\mu\,.t_0, \ldots, \mu\,.t_{N-1}) \in \mathfrak{M}[\![R]\!].\mathsf{s}$
2. $\mathfrak{M}, \mathsf{s}, \mu \vDash \mathrm{K}_a\phi$ iff for every $\mathsf{t} \in \mathsf{S}$, if $\mathsf{s} \sim_a \mathsf{t}$ then $\mathfrak{M}, \mathsf{t}, \mu \vDash \phi$
3. $\mathfrak{M}, \mathsf{s}, \mu \vDash \forall\, x \bullet \phi$ iff $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for every $\nu = \mu \oplus \{x \mapsto a\}$ where $a \in \mathsf{D}_c$

4. $\mathfrak{M}, \mathsf{s}, \mu \vDash \exists\, x \bullet \phi$ iff $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for some $\nu = \mu \oplus \{x \mapsto a\}$ where $a \in \mathsf{D}_c$
5. $\mathfrak{M}, \mathsf{s}, \mu \vDash (\lambda X \bullet \Phi).t$ iff t designates at s and $\mathfrak{M}, \mathsf{s}, \nu \vDash \Phi$ for $\nu = \mu \oplus \{X \mapsto \mathfrak{M}_\mu[\![t]\!].\mathsf{s}\}$

3.1.4 Equality, visibility, and KV

We assume the existence of an equality relation on the object domain D_o. We define a relation of type (I, I) denoted $\doteq$ by

$$x \doteq y \;\;\widehat{=}\;\; (\lambda X, Y \bullet X = Y).(x, y). \tag{3.1.1}$$

which reads: the concepts x and y designate the same object.

As for equality, we can associate, to any relation R of type $(O, \ldots, O)$, a relation $\dot{R}$ of type $(I, \ldots, I)$ by

$$x\, \dot{R}\, y \;\;\widehat{=}\;\; (\lambda X, Y \bullet X\; R\; Y).(x, y). \tag{3.1.2}$$

For every $a \in \mathcal{A}$, we assumed the existence of a special predicate (unary relation) $\mathit{vis}.a$ on concepts. We have seen that the interpretation of $\mathit{vis}.a$ restricted to the flexible constant concepts entirely defines the relation $\sim_a$. The definition of $\mathit{vis}.a$ extends to terms as follows.

Definition 3.14 (Visibility of terms) Consider a triple $(\mathfrak{M}, \mathsf{s}, \mu)$ and an agent a. The interpretation of vis is extended from the concept symbols to any concept term as follows.

$\mathfrak{M}, \mu, \mathsf{s} \models t \in \mathit{vis}.a$ iff t designates at s and for any $\mathsf{s}', \mathsf{s}''$ with $\mathsf{s}' \sim_a \mathsf{s}''$
if t designates at s' then t designates at s'' and $\mathfrak{M}[\![t]\!].\mathsf{s}'' = \mathfrak{M}[\![t]\!].\mathsf{s}'$

We note that in this definition, the quantification takes any two worlds $\mathsf{s}', \mathsf{s}''$. If a concept is visible to an agent a, then it is visible in every world where it designates. We use the following abbreviation for the visibility of a set A of agents

$$t \in \mathit{vis}.A \;\;\widehat{=}\;\; \bigwedge \{a : A \bullet t \in \mathit{vis}.a\}\,.$$

Proposition 3.1 (Visibility of rigid terms) *If a term t is rigid – $\mathfrak{M}[\![t]\!]$ is a constant function on S – then we have, for every agent a, $t \in \mathit{vis}.a$. If f is a* *function of type* $(I, \ldots, I)$*, and if* $(t_0, \ldots, t_{N-1}) \in vis.a$*, then* $f(t_0, \ldots, t_{N-1}) \in vis.a$*.*

Proof.

$$\begin{aligned} \text{If} \qquad \mathfrak{M}[\![f]\!].\mathsf{s} &= \mathfrak{M}[\![f]\!]\mathsf{t}, \\ \mathfrak{M}[\![t_0]\!].\mathsf{s} &= \mathfrak{M}[\![t_0]\!]\mathsf{t}, \\ &\vdots \\ \mathfrak{M}[\![t_{N-1}]\!].\mathsf{s} &= \mathfrak{M}[\![t_{N-1}]\!]\mathsf{t} \end{aligned}$$

then by Definition 3.10

$$\mathfrak{M}[\![f.(t_0, \ldots, t_{N-1})]\!].\mathsf{s} = \mathfrak{M}[\![f.(t_0, \ldots, t_{N-1})]\!]\mathsf{t}.$$

$\square$

The kind of modal formula that we have seen so far is based on the knowledge modality K. Literally a modal formula $\mathrm{K}\Phi$ means “knowing that Φ ”. In some type of systems, we would like to have another modality KV to describe “knowing something” in addition to K which describes “knowing some facts”. In the following, we use the standard notation where K “knowing that” and KV “knowing what”.

Definition 3.15 We define the truth of KV t for a term t when given $(\mathfrak{M}, \mathsf{s}, \mu)$ as follows

$$\begin{aligned} \mathfrak{M}, \mathsf{s}, \mu \models \mathrm{KV}_a t \quad \text{iff} \quad & t \text{ designates at } \mathsf{s} \text{ for } \mu \text{ and} \\ & \text{for any } \mathsf{t} \in \mathsf{S} \text{ if } \mathsf{s} \sim_a \mathsf{t} \text{ then } \mathfrak{M}_\mu[\![t]\!].\mathsf{s} = \mathfrak{M}_\mu[\![t]\!].\mathsf{t} \end{aligned}$$

We would like an expression for KV in terms of other operators as that will simplify our study. If we had allowed quantification over objects (the corresponding semantics can be seen in Appendix A), the definition of $\mathrm{KV}_a x$ would be $\exists\, Y \bullet (\mathrm{K}_a(\lambda X \bullet X = Y)).x$. Semantically, $\mathrm{KV}_a x$ would hold at $\mathfrak{M}, \mathsf{s}$ if there is an object that agent a knows to be what x designates at s. Because objects do not vary from world to world, the object will be the same for all possible worlds in the same $\sim_a$-equivalence class.

We would like a definition of KV using only quantification over concepts. An attempt at this definition is $\exists\, y \bullet \mathrm{K}(x \doteq y)$ but this does not work since any intension x satisfies $\mathrm{K}_a(x \doteq x)$ for any agent. In fact the correct definition requires us to quantify only on the concepts that are visible to the agent.

LAW 3.1 $\mathrm{KV}_a x \;\equiv\; \exists\, y \in \mathit{vis}.a \bullet \mathrm{K}_a(x \doteq y)$

Proof.

$$\mathfrak{M}, \mathsf{s}, \mu \models \exists\, y \bullet y \in \mathit{vis}.a \wedge \mathrm{K}_a(x \doteq y)$$

$\equiv$ *Semantics of* $\exists$

$$\mathfrak{M}, \mathsf{s}, \nu \models y \in \mathit{vis}.a \wedge \mathrm{K}_a(x \doteq y)$$
where $\nu = \mu \oplus \{y \mapsto c\}$ for some $c \in \mathsf{D}_c$

$\equiv$ *Semantics of* $\wedge$

$\mathfrak{M}, \mathsf{s}, \nu \models y \in \mathit{vis}.a$ and $\mathfrak{M}, \mathsf{s}, \nu \models \mathrm{K}_a(x \doteq y)$
where $\nu = \mu \oplus \{y \mapsto c\}$ for some $c \in \mathsf{D}_c$

$\equiv$ *Semantics of* $\mathit{vis}.a$ *and* K_a

For any $\mathsf{t} \sim_a \mathsf{s}, \nu.y.\mathsf{s} = \nu.y.\mathsf{t}$ and for any $\mathsf{t} \sim_a \mathsf{s} \bullet$
$\mathfrak{M}, \mathsf{t}, \nu \models x \doteq y$ where $\nu = \mu \oplus \{y \mapsto c\}$ for some $c \in \mathsf{D}_c$

$\equiv$ *Definition of* $\doteq$
swapping of "and" and "for any"

for any $\mathsf{t} \sim_a \mathsf{s}$, $\nu.y.\mathsf{s} = \nu.y.\mathsf{t}$ and $\mathfrak{M}, \mathsf{t}, \nu \models (\lambda X, Y \bullet X = Y)(x, y)$
where $\nu = \mu \oplus \{y \mapsto c\}$ for some $c \in \mathsf{D}_c$

$\equiv$ *Semantics of* λ

for any $\mathsf{t} \sim_a \mathsf{s}$, $\nu.y.\mathsf{s} = \nu.y.\mathsf{t}$ and $\mathfrak{M}, \mathsf{t}, \theta \models X = Y$
where $\nu = \mu \oplus \{y \mapsto c\}$ for some $c \in \mathsf{D}_c$
and $\theta = \nu \oplus \{X \mapsto \nu.x.\mathsf{t},\; Y \mapsto \nu.y.\mathsf{t}\}$

$\equiv$ *Truth of an atomic formula*

for any $\mathsf{t} \sim_a \mathsf{s}$, $(c.\mathsf{s} = c.\mathsf{t}) \wedge (c.\mathsf{t} = \mu.x.\mathsf{t})$for some $c \in \mathsf{D}_c$

$\equiv$ $\mathsf{s} \sim_a \mathsf{s}$

for any $\mathsf{t} \sim_a \mathsf{s}$, $(\mu.x.\mathsf{s} = c.\mathsf{s}) \wedge (c.\mathsf{s} = c.\mathsf{t}) \wedge (c.\mathsf{t} = \mu.x.\mathsf{t}))$for some $c \in \mathsf{D}_c$

$\equiv$ *transitivity of* $=$ *between objects*

for any $\mathsf{t} \sim_a \mathsf{s}$, $(\mu.x.\mathsf{s} = \mu.x.\mathsf{t})$

$\equiv$ *Definition of* KV_a

$$\mathfrak{M}, \mathsf{s}, \mu \models \mathrm{KV}_a x$$

$\square$

Remark 3.2 (*vis* and KV) We note the difference between $t \in \mathit{vis}.a$ and $\mathrm{KV}_a t$. The first is a property of the term t. The predicate $\mathit{vis}.a$ does not depend on worlds. However $\mathrm{KV}_a t$ can vary from world to world. Therefore we have as a theorem $t \in \mathit{vis}.a \to \mathrm{KV}_a t$ but the converse may not hold. For example, in a card game, if v is the card of player a, then we have $v \in \mathit{vis}.a$ in every world where a holds a card. In a world where another player b can guess the card v, we have $\mathrm{KV}_b v$ whilst $v \notin \mathit{vis}.b$. In a world where player a does not hold any card, v does not designate therefore $v \in \mathit{vis}.a$ is false.

3.1.5 *De dicto* and *de re*

Recall the distinction between the two formulas:

$$\mathrm{K}_a((\lambda X \bullet \Phi.X).t) \qquad (\mathit{de\ re})$$

$$(\lambda X \bullet \mathrm{K}_a(\Phi.X)).t \qquad (\mathit{de\ dicto})$$

These two formulas become equivalent, together with $(\lambda X \bullet (\Phi.X)).t$, when t is rigid. Whilst that might be a limitation in a general modal logic setting, it is not a problem in our setting because we assume ideal agents.

A weaker condition for the equivalence of these formulas was observed by Fitting and Mendelsohn [16]. At a possible world s these two formula are equivalent if the term t designates the same object in all other possible worlds accessible from s vis $\sim_a$. That means in our terms, that $\exists\, x \in \mathit{vis}.a \bullet \mathrm{K}_a(t \doteq x)$ ($\mathrm{KV}_a t$). It was shown in [16] that, for locally rigid terms, the distinction between *de dicto* and *de re* vanishes. Our definition of visibility matches the definition of local rigidity.

Proposition 3.2 (*De dicto* knowledge and visible terms) *If* $t \in \mathit{vis}.a$ *then*

$$\mathrm{K}_a((\lambda X \bullet \Phi).t) \Leftrightarrow (\lambda X \bullet \mathrm{K}_a \Phi).t \Leftrightarrow (\lambda X \bullet \Phi).t$$

Using Fitting's formula for local rigidity [15], we can write *vis* and KV in terms of the other operators

$$(\lambda X \bullet (\mathrm{K}_a(\lambda Y \bullet X = Y).t)).t \Leftrightarrow t \in \mathit{vis}.a \tag{3.1.3}$$

$$\exists\, x \bullet (\lambda X \bullet (\mathrm{K}_a(\lambda Y \bullet X = Y).t)).x \Leftrightarrow \mathrm{KV}_a t \tag{3.1.4}$$

3.1.6 Common Knowledge

We have defined the knowledge of a concept t by an agent a with the formula $\exists\, x \in \mathit{vis}.a \bullet \mathrm{K}_a(t \doteq x)$ (which we abbreviate to $\mathrm{KV}_a t$ when it is convenient). In this definition, $\mathit{vis}.a$ is a rigid relation. Similarly, we define a formula for the common knowledge of a term t by agents in a group A (abbreviated to CVt) with

$$CVt = \exists\, x \in \mathit{vis}.A \bullet \bigwedge \{a : A \bullet \mathrm{K}_a(v \doteq x)\}\,. \tag{3.1.5}$$

We note that

$$x \in \mathit{vis}.A = \bigwedge \{a : A \bullet x \in \mathit{vis}.a\}$$

and

$$E_A(v \doteq x) = \bigwedge \{a : A \bullet \mathrm{K}_a(v \doteq x)\}\,.$$

In the definition of the common knowledge of a term, the required concept visible to everyone is unique. In contrast in the formula "everyone knows the concept t", the required visible concept may vary for each agent.

$$EVt = \bigwedge \{a : A \bullet\ \exists\, x \in \mathit{vis}.a \mid \mathrm{K}_a(v \doteq x)\}\,.$$

3.1.7 Axioms and theorems

An axiomatisation of FOIL was given in [14] and is included in the Appendix A.1.2 with some additional theorems. The axioms are for a basic modal logic (axiom K) with one modality and an assumed equality $=$ between objects. For reasoning about the knowledge of ideal agents, we need to add to these, the axioms $(T, K4, K5)$ in Section 1.4. The following theorems will also be used in some of our case studies.

Law 3.2 $\mathrm{K}_a(u \doteq v) \wedge \mathrm{K}_a(v \doteq w) \Rightarrow \mathrm{K}_a(u \doteq w)$

Law 3.3 $\mathrm{KV}p \wedge \mathrm{K}(p \vee q) \Rightarrow \quad p \vee \mathrm{K}q$

Law 3.4 $\mathrm{KV}p \wedge \mathrm{K}(p \wedge q) \Rightarrow \neg p \vee \mathrm{K}q$

Law 3.5 $p \in \mathit{vis} \vdash \mathrm{K}(p \vee q) \Leftrightarrow \quad p \vee \mathrm{K}q$

LAW 3.6 $p \in vis \vdash \mathrm{K}(p \wedge q) \Leftrightarrow \neg p \vee \mathrm{K}q$

Proof of Law 3.3 . Suppose $p \in vis$, then

$$\mathrm{K}(p \vee q)$$

$\Leftrightarrow$

$$\mathrm{K}(\neg p \rightarrow q)$$

$\Leftrightarrow$ $\quad$ *true* $= p \vee \neg p$

$$\left(\vee \begin{array}{c} p \wedge \mathrm{K}(\neg p \rightarrow q) \\ \neg p \wedge \mathrm{K}(\neg p \rightarrow q) \end{array}\right)$$

$\Leftrightarrow$ $\quad$ $p \in vis \;\Rightarrow\; p \leftrightarrow \mathrm{K}p$ *(Prop.3.2)*

$$\left(\vee \begin{array}{c} \mathrm{K}p \wedge \mathrm{K}(\neg p \rightarrow q) \\ \mathrm{K}\neg p \wedge \mathrm{K}(\neg p \rightarrow q) \end{array}\right)$$

$\Leftrightarrow$ $\quad$ $\Rightarrow$: *Axiom D* $\quad$ $\Leftarrow$: $\mathrm{K}q \rightarrow \mathrm{K}(q \vee p)$

$$\left(\vee \begin{array}{l} \mathrm{K}p \wedge \mathrm{K}(p \vee q) \\ \mathrm{K}\neg p \wedge \mathrm{K}q \end{array}\right)$$

$\Leftrightarrow$ $\quad$ $\Leftarrow$: $\mathrm{K}p \rightarrow \mathrm{K}(q \vee p)$

$$\left(\vee \begin{array}{l} \mathrm{K}p \\ \mathrm{K}\neg p \wedge \mathrm{K}q \end{array}\right)$$

$\Leftrightarrow$ $\quad$ $p \in vis \;\Rightarrow\; p \leftrightarrow \mathrm{K}p$ *(Prop. 3.2)*

$$\left(\vee \begin{array}{l} p \\ \neg p \wedge \mathrm{K}q \end{array}\right)$$

$\Leftrightarrow$ $\quad$ *Distributing* $\vee$

$$p \vee \mathrm{K}q$$

□

3.2 Relations between models

We are interested in the relations between concrete models over the same object domain D_o and having finite set of attributes.

3.2.1 Modal equivalence and isomorphism

Recall that, in propositional modal logic, models satisfy the same set of modal sentences if and only if there is a bisimulation between them. The concrete models that we consider share a unique domain D_o and they are defined such that the worlds and the accessibility relations are coded into the interpretations. We have the following.

Definition 3.16 (Theory) Given a vocabulary $\mathcal{V}$, the theory of a $\mathcal{V}-$model $\mathfrak{M}$ is

$$Th_{\mathcal{V}}(\mathfrak{M}) = \{\mathcal{V}\text{-sentences } \phi \text{ such that } \mathfrak{M} \models \phi\}$$

and for any subset $\mathcal{V}'$ of $\mathcal{V}$

$$Th_{\mathcal{V}'}(\mathfrak{M}) = \{\mathcal{V}'\text{-sentences } \phi \text{ such that } \mathfrak{M} \models \phi\}$$

Proposition 3.3 *Let a vocabulary $\mathcal{V} = (\mathcal{I}_f, \mathcal{I}_r, \mathcal{F}, \mathcal{R})$ such that $\mathcal{I}_f$ is finite. Two concrete $\mathcal{V}$-models $\mathfrak{M}$ and $\mathfrak{M}'$, having the same object domain D_o, are modally equivalent if and only if they are isomorphic.*

$$Th_{\mathcal{V}}(\mathfrak{M}) = Th_{\mathcal{V}}(\mathfrak{M}') \text{ iff } \mathfrak{M} \equiv \mathfrak{M}' \tag{3.2.1}$$

Proof. The reverse implication is straightforward.

We will prove the forward implication. Consider two modally equivalent models $\mathfrak{M} = \langle \mathsf{S}, (\sim_a)_{a\in\mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}[\![\,.\,]\!]\rangle$ and $\mathfrak{M} = \langle \mathsf{S}', (\sim'_a)_{a\in\mathcal{A}}, \mathsf{D}_o, \mathsf{D}'_c, \mathfrak{M}'[\![\,.\,]\!]\rangle$, i.e., $\mathfrak{M}$ and $\mathfrak{M}'$ validate the same set of modal formulas. In particular they satisfy the same set of classical formulas. The rigid concept domains are the constant total functions to D_o thus isomorphic to D_o for both models. There is an isomorphism $\mathfrak{F}$ between the rigid structures $\langle \mathsf{D}_o, \mathcal{V}_r \lhd \mathfrak{M}[\![\,.\,]\!]\rangle$ and $\langle \mathsf{D}'_o, \mathcal{V}_r \lhd \mathfrak{M}'[\![\,.\,]\!]\rangle$ where $\mathcal{V}_r = (\mathcal{I}_r, \mathcal{F}, \mathcal{R})$. We will prove that these rigid components augmented with the set of possible worlds remain isomorphic and so will be the two models.

For simplicity, we assume only two symbols u, v in the $\mathcal{I}_f$ set of attributes. The proof can be extended for finitely many symbols. A possible world $\mathsf{s} \in \mathsf{S}$ is either a total function of the form $(u, v) \mapsto (\mathsf{N}, \mathsf{M})$ or a partial function of the form $u \mapsto \mathsf{N}$. We will prove that $\langle \mathsf{S}, \mathsf{D}_o, \mathcal{V}_r \lhd \mathfrak{M}[\![\,.\,]\!]\rangle$ and $\langle \mathsf{S}', \mathsf{D}'_o, \mathcal{V}_r \lhd \mathfrak{M}'[\![\,.\,]\!]\rangle$

are isomorphic by showing that

$$\begin{aligned} (u,v) \mapsto (\mathsf{N},\mathsf{M}) \in \mathsf{S} &\quad\equiv\quad (u,v) \mapsto (\mathfrak{F}.\mathsf{N},\mathfrak{F}.\mathsf{M}) \in \mathsf{S}' \\ \text{and } (u \mapsto \mathsf{N}) \in \mathsf{S} &\quad\equiv\quad (u \mapsto \mathfrak{F}.\mathsf{N}) \in \mathsf{S}'. \end{aligned}$$

We denote by n, m the rigid concepts that interpret rigidly to N, M

$(u,v) \mapsto (\mathsf{N},\mathsf{M}) \notin \mathsf{S}$ (resp. $u \mapsto \mathsf{N}$)

$\equiv$ *Definition of* S

for any s in S: not ($\mathsf{s}.u = \mathsf{N}$ and $\mathsf{s}.v = \mathsf{M}$)
(resp. $\mathsf{s}.u = \mathsf{N}$ and $v \notin \operatorname{dom} \mathsf{s}$)

$\equiv$ *Definition of* $\mathfrak{M}$

for any s in S: not ($\mathfrak{M}[\![u]\!].\mathsf{s} = \mathfrak{M}[\![n]\!].\mathsf{s}$ and $\mathfrak{M}[\![v]\!].\mathsf{s} = \mathfrak{M}[\![m]\!].\mathsf{s}$)
(resp. not ($\mathfrak{M}[\![u]\!].\mathsf{s} = \mathfrak{M}[\![n]\!].\mathsf{s}$ and $\mathsf{s} \notin \mathfrak{M}[\![v]\!]$))

$\equiv$ *Truth in* $\mathfrak{M}$

for any s in S: $\mathfrak{M}, \mathsf{s} \models \neg(u = n \wedge v = m)$
(resp. $\mathfrak{M}, \mathsf{s} \models \neg(u = n \wedge \neg D.v)$)

$\equiv$ *Modal equivalence of* $\mathfrak{M}$ *and* $\mathfrak{M}'$

for any t in S': $\mathfrak{M}', \mathsf{t} \models \neg(u = n \wedge v = m)$
(resp. $\mathfrak{M}', \mathsf{t} \models \neg(u = n \wedge \neg D.v)$)

$\equiv$ *Truth in* $\mathfrak{M}'$

for any t in S': not ($\mathfrak{M}'[\![u]\!].\mathsf{t} = \mathfrak{M}'[\![n]\!].\mathsf{t}$ and $\mathfrak{M}'[\![v]\!].\mathsf{t} = \mathfrak{M}'[\![m]\!].\mathsf{t}$)
(resp. not ($\mathfrak{M}'[\![u]\!].\mathsf{t} = \mathfrak{M}'[\![n]\!].\mathsf{t}$ and $\mathsf{t} \notin \mathfrak{M}[\![v]\!]$)

$\equiv$ *Definition of* $\mathfrak{M}'$
and isomorphism $\mathfrak{F}$

for any t in S': not ($\mathsf{t}.u = \mathsf{N}$ and $\mathsf{t}.v = \mathsf{M}$)
resp. (not ($\mathsf{t}.u = \mathfrak{F}.\mathsf{N}$ and $v \notin \mathsf{t}$))

$\equiv$ *Definition of* S'

$(u,v) \mapsto (\mathfrak{F}.\mathsf{N},\mathfrak{F}.\mathsf{M}) \notin \mathsf{S}'$ (resp. $u \mapsto \mathfrak{F}.\mathsf{N} \notin \mathsf{S}'$)

The other components: $\mathsf{D}_c \subseteq \mathsf{S} \nrightarrow \mathsf{D}_o$ and $(\sim_a)_{a \in \mathcal{A}}$. The isomorphism between D_c and D_c is obtained in a similar way as above but considering a formula of the form $(u = n \wedge v = m) \Rightarrow (i = c \vee \neg D.I)$ for an intension i designating the same object as some rigid intension c. The equivalences are defined from $(vis.a)$ on the set of possible worlds. But the relation $vis.a$ is rigid and the two sets of worlds are isomorphic, so the accessibility relations will be isomorphic. □

The previous theorem states the modal equivalence of two models defined on the same vocabulary. The following lemma states that, if we add only an attribute that is invisible to every agent into a model, then, the resulting model validates the same set of sentences (not containing the new attribute symbols).

LEMMA 3.1 *Assume a $\mathcal{V}$-model $\mathfrak{M}$ and let $\mathfrak{M}'$ be an extension of $\mathfrak{M}$ with the attribute w i.e., $(\{w\} \lhd W', \mathcal{F}, \mathcal{R}, \mathcal{V} \lhd \mathfrak{M}') = \mathfrak{M}$. If for any $a \in \mathcal{A}$ $w \notin vis.a$, then*

$$Th_{\mathcal{V}}(\mathfrak{M}) = Th_{\mathcal{V}}(\mathfrak{M}'). \tag{3.2.2}$$

Proof. Adding a new attribute w to a model can change only its accessibility relations. Thus adding an attribute hidden to every agent does not change the set sentences (not having w) validated by the model. □

A similar result holds also when adding an attribute that is constant across the possible worlds.

LEMMA 3.2 *Adding an attribute that has the same value in all the possible worlds does not change the set of valid sentences (not containing the new attribute symbols), independently of the visibility of the new attribute.*

3.2.2 Standard model and concrete models

THEOREM 3.1 *For any standard $\mathcal{V}$-model $\mathfrak{M}$, we can find a concrete $\mathcal{V}'$-model $\mathfrak{M}'$ with $\mathcal{V}' \supseteq \mathcal{V}$ that validates the same $\mathcal{V}$-sentences i.e.,*

$$Th_{\mathcal{V}}(\mathfrak{M}) = Th_{\mathcal{V}}(\mathfrak{M}') \tag{3.2.3}$$

Proof. Take "the name of the world" to be a new constant concept that is not visible to any agent. □

REMARK 3.3 (Graphical representation of a concrete model) We do not specify what is the actual world in our concrete models. When reasoning about

$$\boxed{\begin{array}{c} \langle v_0 \mapsto \mathsf{m}_0 \rangle_{A_0} \\ \vdots \\ \langle v_N \mapsto \mathsf{m}_N \rangle_{A_{N-1}} \end{array}} \quad \text{or} \quad \boxed{\begin{array}{c} \langle \mathsf{m}_0 \rangle_{A_0} \\ \vdots \\ \langle \mathsf{m}_N \rangle_{A_{N-1}} \end{array}}$$

FIGURE 3.1 Graphical representations of a state given attributes $(v_0, \ldots, v_{N-1})$. Each m_i is an object in the domain of the model. The labelled brackets $\langle\rangle_{A_i}$ indicate that the attribute inside them is not visible to agents in A_i, i.e., $v_i \in \mathit{hid}.A_i$. The simpler version on the right is used when there is no ambiguity.

programs, the definition of refinement between programs does not depend on the initial worlds. Thus, we do not define explicitly an actual state, most of the reasoning is on the actual state of knowledge.

Assume a set of tuple of attributes $(v_0, \ldots, v_N)$. A possible world is defined by associating a tuple $(\mathsf{m}_0, \ldots, \mathsf{m}_N)$ of objects to the attributes. And a model is defined by giving a set of states and associating to each attribute v_i the set $\mathit{mask}.v_i$ of agents that cannot see v_i

$$\mathit{mask}.v_i = \{a : \mathcal{A} \mid v_i \notin \mathit{vis}.a\}.$$

EXAMPLE 3.3 (Three wise men puzzle) *In this example, we consider three agents wearing a hat that can be black or white. Each agent can see the hat of the two others but not his own.*

To construct a concrete model for this situation, let $a_{i=0,1,2}$ be the three agents, h_i the color of a_i's hat, and $\{\mathsf{b}, \mathsf{w}\}$ be the set of colors black and white. A possible world is determined by the color of each agent's hat. The situation is formalised: $h_j \notin \mathit{vis}.a_i$ iff $j = i$.

- *The world attributes are $\{h_0, h_1, h_2\}$.*
- *The possible worlds are the elements of $\{h_0, h_1, h_2\} \to \{\mathsf{b}, \mathsf{w}\}^3$.*
- *The visibility relations are determined by $\mathit{mask}.v_i = \{a_i\}$.*

We note that a possible world is a total function because each agent wears a hat.

From Figure 3.2, we observe that every possible world is connected to

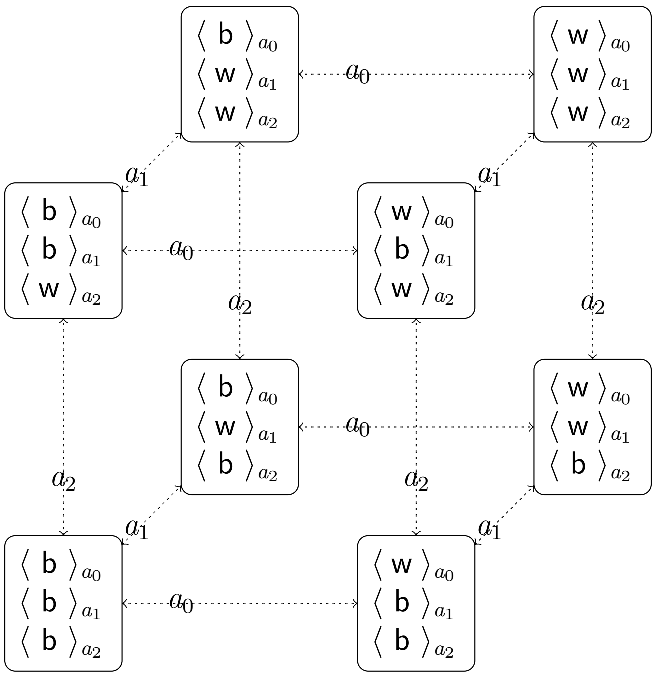

FIGURE 3.2 Concrete epistemic model for the three wise men a_0, a_1, a_2. Each node is a possible world and is determined by the hat's color h_0, h_1, h_2 (these are the world attributes). Invisibility of h_i by $a_i - mask.h_i = \{a_i\}$ is represented by putting brackets $\langle\rangle_{a_i}$ around the value of h_i. An arrow $\longleftrightarrow^{a_i}$ is put between two worlds if they differ only on what is inside $\langle\rangle_{a_i}$.

another possible world by an arrow labelled with a_i (for every i). This means that no wise man knows the colour of his hat. The use of a concrete epistemic model instead of the Shadow Model for reasoning about computation (as we shall see in Chapter 4) allows us to use well-established laws and techniques of epistemic logic, for instance the drawing of possible worlds.

3.2.3 Update of a concrete model

Suppose that we would like a change of information in a model i.e., we do not change any world attribute inside the worlds. *What are the possible ways to transform a concrete model like the cube in Figure 3.2?* An attempt to change the model results in giving information to a particular agent, thus removing some of his arrows of ignorance. But removing or adding arrows (accessibility) relations cannot be done directly; these will not give us another concrete model.

Because in a concrete model, arrows are encoded into the states. In fact, in Figure 3.2, arrows were only drawn for the sake of presentation.

Removing a state or a group of states corresponds exactly to a public announcement; see Section 1.5. The only possible update of a concrete model into another concrete model is a public announcement unless we make some factual change in the model. In fact the factual change need not change the values of the attributes present in the models. If, only information has been given, then more attributes are added to the possible worlds. To model a private announcement, we encode the given information into some private resource that are added to the set of attributes.

3.3 First-order public announcement logic

Definition

The Public Announcement Logic (PAL) defines two formula constructs $\langle\psi\rangle\phi$ and $[\psi]\phi$. Originally, the underlying logic is propositional. We will give a first-order version of it. The truth definition of a PAL formula in a model is the same as its original version, and is taken directly from the reading of the formula.

- The formula $\langle\psi\rangle\phi$ reads as "after some announcement of ψ, ϕ holds."
- The formula $[\psi]\phi$ reads as "after every announcement of ψ, ϕ holds."

We have the duality

$$[\phi]\psi \Leftrightarrow \neg\langle\phi\rangle\neg\psi. \tag{3.3.1}$$

Semantics

The following interpretation of the formula $\langle\psi\rangle\phi$ in a possible model is common in the literature on PAL. However, we interpret the formula in a first-order model.

Definition 3.17 The truth of the formula $\langle\psi\rangle\phi$ given a triple $(\mathfrak{M}, \mathsf{s}, \mu)$ is given by

$$\mathfrak{M}, \mathsf{s}, \mu \models \langle\psi\rangle\phi \ \equiv \ (\mathfrak{M}, \mathsf{s}, \mu \models \psi) \text{ and } (\mathfrak{M}_{|\psi}, \mathsf{s}, \mu \models \phi)$$

where $\mathfrak{M}_{|\psi} = \langle \mathsf{S}_{|\psi}, (\sim_{a|\psi})_{a\in\mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}_{|\psi}[\![\,.\,]\!]\rangle$ such that

$$\mathsf{S}_{|\psi} \;=\; \{\mathsf{s} : \mathsf{S} \mid \mathfrak{M}, \mathsf{s} \models \psi\}$$

$$\sim_{a|\psi} = \sim_a \cap (\mathsf{S}_{|\psi} \times \mathsf{S}_{|\psi})$$

$$\mathfrak{M}_{|\psi}[\![\,.\,]\!] \;=\; \mathsf{S}_{|\psi} \lhd \mathfrak{M}[\![\,.\,]\!]$$

The truth definition for $[\psi]\Phi$ is similar but with an implication instead of a conjunction.

Axioms

Existing PAL axiom systems that we know are for propositional PAL. In the following, we give the axioms reducing the first-order PAL formulas into first-order epistemic formulas. Most axioms are proved, similarly to the propositional case ([11]), using the update model definition of PAL. Most axiomatisations of PAL in the literature do not involve KV except in its perhaps original form [33], although it was not totally axiomatised. The use of quantifiers lets us give a direct PAL axiom for KV.

Proposition 3.4 *For atomic formula $\alpha = R(t_0, \ldots, t_{N-1})$ and a formula ψ in which x is not free, we have:*

$$\langle\psi\rangle\alpha \;\Leftrightarrow\; \psi \wedge \alpha \tag{3.3.2}$$

$$\langle\psi\rangle\neg\phi \;\Leftrightarrow\; \psi \wedge \neg\langle\psi\rangle\phi \tag{3.3.3}$$

$$\langle\psi\rangle\phi_0 \wedge \phi_1 \;\Leftrightarrow\; \langle\psi\rangle\phi_0 \wedge \langle\psi\rangle\phi_1 \tag{3.3.4}$$

$$\langle\psi\rangle\phi_0 \vee \phi_1 \;\Leftrightarrow\; \langle\psi\rangle\phi_0 \vee \langle\psi\rangle\phi_1 \tag{3.3.5}$$

$$\langle\psi\rangle\phi_0 \rightarrow \phi_1 \;\Leftrightarrow\; \psi \wedge (\langle\psi\rangle\phi_0 \rightarrow \langle\psi\rangle\phi_1) \tag{3.3.6}$$

$$\langle\psi\rangle\,\exists\, x\, \phi \;\Leftrightarrow\; \exists\, x \bullet \langle\psi\rangle\phi \tag{3.3.7}$$

$$\langle\psi\rangle\,\forall\, x \bullet \phi \;\Leftrightarrow\; \forall\, x \bullet \langle\psi\rangle\phi \tag{3.3.8}$$

$$\langle\psi\rangle(\lambda X \bullet \phi).t \;\Leftrightarrow\; (\lambda X \bullet \langle\psi\rangle\phi).t \tag{3.3.9}$$

$$\langle\psi\rangle\mathrm{K}\phi \;\Leftrightarrow\; \psi \wedge \mathrm{K}(\psi \rightarrow \langle\psi\rangle\phi) \tag{3.3.10}$$

$$\langle\psi\rangle\mathrm{KV}_a t \;\Leftrightarrow\; \psi \wedge (\exists\, y \in \mathit{vis}.a \bullet \mathrm{K}_a(\psi \rightarrow t \doteq y)) \tag{3.3.11}$$

Proposition 3.5 *For atomic formula $\alpha = R(t_0, \ldots, t_{N-1})$ and a formula ψ*

in which x is not free, we have:

$$[\psi]\alpha \Leftrightarrow \psi \to \alpha \tag{3.3.12}$$
$$[\psi]\neg\phi \Leftrightarrow \psi \to \neg[\psi]\phi \tag{3.3.13}$$
$$[\psi]\phi_0 \wedge \phi_1 \Leftrightarrow [\psi]\phi_0 \wedge [\psi]\phi_1 \tag{3.3.14}$$
$$[\psi]\phi_0 \vee \phi_1 \Leftrightarrow [\psi]\phi_0 \vee [\psi]\phi_1 \tag{3.3.15}$$
$$[\psi]\phi_0 \to \phi_1 \Leftrightarrow [\psi]\phi_0 \to [\psi]\phi_1 \tag{3.3.16}$$
$$[\psi]\,\exists\, x \bullet \phi \Leftrightarrow \exists\, x \bullet [\psi]\phi \tag{3.3.17}$$
$$[\psi]\,\forall\, x \bullet \phi \Leftrightarrow \forall\, x \bullet [\psi]\phi \tag{3.3.18}$$
$$[\psi](\lambda X \bullet \phi).t \Leftrightarrow (\lambda X \bullet [\psi]\phi).t \tag{3.3.19}$$
$$[\psi]\mathrm{K}\phi \Leftrightarrow \psi \to \mathrm{K}([\psi]\phi) \tag{3.3.20}$$
$$[\psi]\mathrm{KV}_a t \Leftrightarrow \psi \to (\exists\, y \in \mathit{vis}.a \bullet \mathrm{K}_a(\psi \to t \doteq y)) \tag{3.3.21}$$

Following are some properties of PAL listed and proved to be sound in [11].

Proposition 3.6

$$\langle\psi\rangle\phi \Leftrightarrow \psi \wedge [\psi]\phi \tag{3.3.22}$$
$$\langle\psi\rangle\phi \Leftrightarrow \psi \wedge \langle\psi\rangle\phi \tag{3.3.23}$$
$$[\psi]\phi \Leftrightarrow \psi \to [\psi]\phi \tag{3.3.24}$$
$$[\psi]\phi \Leftrightarrow \psi \to \langle\psi\rangle\phi \tag{3.3.25}$$

Proposition 3.7 *More properties of PAL.*

$$\langle\psi_0\rangle\langle\psi_1\rangle\phi \Leftrightarrow \langle\langle\psi_0\rangle\psi_1\rangle\phi \tag{3.3.26}$$
$$[\psi_0][\psi_1]\phi \Leftrightarrow [\langle\psi_0\rangle\psi_1]\phi \Leftrightarrow [\psi_0 \wedge [\psi_0]]\phi \tag{3.3.27}$$

Soundness of (3.3.27) can be found in [11] and that of (3.3.26) is found in [33].

Summary

We have developed a first-order epistemic logic based on First-Order Intensional Logic (FOIL) [13] by Fitting; the version in [14] is closer to the one presented here. Other versions of FOIL is found in [16] and in [7]. We have adapted FOIL to suit our own notation, intension or individual concepts are simply called concepts, constant concepts and functions are considered, and for every object an associated (rigid) concept was assumed, thus quantification was considered only over concepts.

Another difference is in the definition of models. We define the possible worlds with the set of nonrigid constants, and the accessibility relations with the visibility of these nonrigid constants by the agents. The resulting model is a multidimensional frame. It is equivalent to a Kripke structure. While a Kripke structure is defined by a set of possible states, and sets of relations on the possible states $(W, (R_i)_i)$, the relations in a multidimensional frame is "coded" in its states.

> "Multi-Dimensional Modal Logic is about special relational structures, in which the states (or possible states), rather than being abstract entities, have some inner structure; furthermore, the accessibility relations between states are determined by this inner structure. Marx and Venema in *Multi-Dimensional Modal Logic* ([22])"

The first-order epistemic logic was extended to a public announcement logic. The use of quantifiers allows us to define formulas such as $\mathrm{KV}_a v$ which reads "a knows the value of v" (in addition to the classical modality $\mathrm{K}_a\phi$ –"a knows that ϕ").

Chapter 4

Program semantics

This chapter presents the programming models that justify our program algebra. The first part of this chapter treats the denotational semantics sensitive to information flow. The computational models used in classical programs are not suitable for reasoning about knowledge. This is due to the fact that in the classical models a logical property (some facts about the program variables) is a property of individual states. In reasoning about information flow, a logical property (some facts or some information that an agent knows about the program variables) is a property of a set of states and cannot always be validated by looking at individual states.
In the second part of this chapter, we propose a weakest precondition semantics extending those given by Morgan [28]. Our wp-semantics will make use of the first-order Public Announcement Logic developed in Chapter 3. In the last section, we give the connection between the two semantics for some of the commands in our programming language.

4.1 Denotational semantics

To reason about programs sensitive to information flow, Morgan [28] proposed his *Shadow Model* to replace the classical relational model. Traditional states are determined by the values of the program variables. A state in the shadow model is also determined by the other possible values of the secret variable according to the attacker.

Example 4.1 *Consider a public variable v and a secret variable h. A classical state is a couple of values* v, h, *and a state can validate or not any first-order formulas on v, h. For example the classical formula $h \in E$ is true at a state where* $\mathsf{h} \in [\![E]\!].\mathsf{v}.\mathsf{h}$. *However, in the Shadow Model, a state is a triple* $\mathsf{v}, \mathsf{h}, \mathsf{H}$ *and the model allows validation of modal formulas. For example, the modal formula $K(h \in E)$ is true at a state* $\mathsf{v}, \mathsf{h}, \mathsf{H}$ *if* $\mathsf{H} \subseteq [\![E]\!].(\mathsf{v}, \mathsf{h}, \mathsf{H})$.

A single agent (the attacker) was assumed in the Shadow Model and reasoning about multiple agents is possible only by taking the point of view of each agent separately. This approach cannot take into account scenarios with nesting of knowledge, and cannot be used for instance to reason about common knowledge. A Shadow model that captures the knowledge of more than one agents would consider states not as triples $\mathsf{v}, \mathsf{h}, \mathsf{H}$ but as tuples $\mathsf{v}, \mathsf{h}, \mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_N$. Each H_i would be the shadow of h according to agent A_i But rather than extending the Shadow model, our approach is to develop a programming model that can serve also as a model for (multi-modal) epistemic logic. We give a more elaborate definition of the state (or epistemic state) of a program. These epistemic states serve both as a computational model and as a model for the logic. This section makes use of the concrete epistemic models presented in Chapter 3.

4.1.1 States

For standard programs, a state is uniquely determined by the values of the program variables. A state is a function from the global variable symbols to the domain: a state is determined by the current values of the global program variable. For our purpose, we need to record not only the current values of global variables but also their past values. The latter are not erased even when the variables are discarded.

Definition 4.1 (State) The state s of a program is a partial function

$$\mathcal{P} \cup \mathcal{H} \nrightarrow \mathsf{D}$$

where $\mathcal{P} = \{v_0, \ldots, v_{N-1}\}$ is a set of global variable symbols and $\mathcal{H} = \{hist.v_0 \cup \ldots \cup hist.v_{M-1}\}$ is a set of historical variable symbols. Each $hist.v_i$ is a list of symbols. The union $\mathcal{P} \cup \mathcal{H}$ is the set of the state attributes.

Example 4.2 *Let P be the program*

$$v :\in \{0,1\} \mathbin{;} \mathsf{var}\ w \bullet w := v\ \mathsf{end} \mathbin{;} v := 3.$$

If P is executed from a state $v \mapsto 1$ then the possible resulting states are

$$\begin{array}{rcl} v & \mapsto & 3 \\ hist.v & \mapsto & (1,0) \\ hist.w & \mapsto & (0) \end{array} \quad \textit{and} \quad \begin{array}{rcl} v & \mapsto & 3 \\ hist.v & \mapsto & (1,1) \\ hist.w & \mapsto & (1) \end{array}$$

We read from these that the current global variable is v and its value is 3 in both possible states. The history of v is either $(1,0)$ or $(1,1)$ corresponding to histories (0) and (1) of w. When w is no longer available (no longer global), only its history remains in the final state.

The state space is determined by the available (global) program variables $\mathcal{P}$, which are a subset of the attributes A. The elements of A outside $\mathcal{P}$, are the variables that are only "part of the history". As defined in the syntax, each program variable in $\mathcal{P}$ has a special type specifying the set of agents that can see it, or equivalently the agents that cannot see it. We shall use the latter for graphical representation. We put brackets $\langle\rangle_{a_i}$ around the value of a variable that agent a_i cannot see.

Example 4.3 *In Example 4.2, consider two variables $vis.av$ and $vis.bw$. The mask of $\mathsf{s}.v$ and every function whose domain is in $hist.v$ is $\{a\}$. The mask of every function with domain in $hist.w$ is $\{b\}$.*

$$\begin{array}{rcl} v & \mapsto & \langle 3\rangle_b \\ hist.v & \mapsto & \langle 1,0\rangle_b \\ hist.w & \mapsto & \langle 0\rangle_a \end{array} \quad \textit{and} \quad \begin{array}{rcl} v & \mapsto & \langle 3\rangle_b \\ hist.v & \mapsto & \langle 1,1\rangle_b \\ hist.w & \mapsto & \langle 1\rangle_a \end{array}$$

4.1.2 Epistemic state

We assume a fixed alphabet of functions, relations, constant rigid concepts symbols $\mathcal{F} \cup \mathcal{R} \cup \mathcal{I}_r$ and their interpretation.

Definition 4.2 (Epistemic state) An *epistemic state* $\mathfrak{M}$ is a set of states with history and visibility. Each epistemic state $\mathfrak{M}$ is determined by its set of attributes $(\mathcal{P} \cup \mathcal{H}_{\mathfrak{M}})$, its interpretation of the attributes, and its interpretation of *vis*.

$$\mathfrak{M} \mathrel{\widehat{=}} ((\mathcal{P} \cup \mathcal{H}_{\mathfrak{M}}) \lhd \mathfrak{M}[\![.]\!], \mathfrak{M}[\![vis]\!]) = (\mathsf{S}, \mathfrak{M}[\![vis]\!])$$

This definition corresponds to the definition of a concrete model of Chapter 3. Recall that, we assume a unique domain of object and a fixed interpretation of the rigid symbols. An accessibility relation $\sim_a$ is defined for each agent a from this model. The flexible concept symbols are $\mathcal{I}_f = \mathcal{P} \cup \mathcal{H}$. We recall also that v designates at s if $v \in \operatorname{dom} \mathsf{s}$ and that $\mathsf{s}.v = \mathfrak{M}[\![v]\!].\mathsf{s}$

$$\begin{aligned}&\mathsf{s} \sim_a \mathsf{t} \text{ if for all } v \text{ in } \mathcal{I}_f \text{ such that } v \in \mathfrak{M}[\![vis.a]\!] :\\ &\qquad\qquad \text{if } v \text{ designates at } \mathsf{s}, \text{ then } v \text{ designates at } \mathsf{t} \text{ and } \mathfrak{M}[\![v]\!].\mathsf{s} = \mathfrak{M}[\![v]\!].\mathsf{t}.\end{aligned}$$

It is possible to simplify the history of a state by preserving only the values that affect the future information flows. But this is difficult in general as it amounts to comparing the epistemic models not containing the values discarded to the original epistemic model. In contrast, the Shadow model discards unnecessary values. The state of knowledge is completely captured by the shadow set (H in the triple $\mathsf{v}, \mathsf{h}, \mathsf{H}$, see Example 4.1). The Shadow semantics puts the description of operations into a programming language but only a simpler logic of knowledge is supported there. Now, we give the semantics of the programming commands using the epistemic model defined above.

4.1.3 Program semantics

The semantics of a program P, denoted $[\![P]\!]$, is a function from an initial information set $\mathfrak{M}_0$ to a final information set $\mathfrak{M}_f$. For some programming commands (most classical commands) $[\![P]\!]$ can be defined on individual states of $\mathfrak{M}_0$, then union of final states give the possible states states $\mathfrak{M}_f$. The semantics of expressions and logical expressions are those given in Chapter 3. When there is no ambiguity we will write $\mathsf{s}[\![v]\!]$ instead of $\mathfrak{M}, \mathsf{s}[\![v]\!]$.

Local variables

Declaring a new variable w extends the state space as w is added to $\mathcal{P}$. The closing of the scope of w moves to the original state space and appends the latest value of w to its history.

$$\begin{aligned}
&[\![\mathsf{vis}_a\ w]\!].\mathsf{s}_0\ = \{n : \mathsf{D} \bullet \mathsf{s}_0 \oplus \{w \mapsto n\}\}\\
&[\![\mathsf{vis}_a\ w]\!].\mathfrak{M}_0 = (\cup \{\mathsf{s}_0 : \mathsf{S}_0 \bullet [\![\mathsf{vis}_a\ w]\!].\mathsf{s}_0\}\ ,\ \mathfrak{M}_0[\![vis]\!] \oplus \{w \mapsto \{a\}\})
\end{aligned}$$

$$\begin{aligned}
&[\![w\ \mathsf{end}]\!].\mathsf{s}_0\ = (\{w\} \lhd \mathsf{s}) \oplus \{hist.w \mapsto \mathsf{s}[\![hist.w]\!] \frown \mathsf{s}[\![w]\!]\})\\
&[\![w\ \mathsf{end}]\!].\mathfrak{M}_0 = (\{\mathsf{s}_0 : \mathsf{S}_0 \bullet [\![w\ \mathsf{end}]\!].\mathsf{s}_0\}, \{w\} \lhd \mathfrak{M}_0[\![vis]\!])
\end{aligned}$$

Assignment

Assigning an expression e to a variable v appends the current value of v to its history $\mathsf{s}[\![hist.v]\!]$ and changes its current value $\mathsf{s}[\![v]\!]$ to $\mathsf{s}[\![e]\!]$

$$\begin{aligned}
&[\![v := e]\!].\mathsf{s}_0\ = \{\mathsf{s}_0 \oplus \{v \mapsto \mathsf{s}_0[\![e]\!], hist.v \mapsto \mathsf{s}_0[\![hist.v]\!] \frown \mathsf{s}_0[\![v]\!]\}\\
&[\![v := e]\!].\mathfrak{M}_0 = (\{\mathsf{s} : \mathfrak{M}_0 \bullet [\![v := e]\!].\mathsf{s}_0\}, \mathfrak{M}_0[\![vis]\!])
\end{aligned}$$

$$\begin{aligned}
&[\![v :\in E]\!].\mathsf{s}_0\ = \{e : E \bullet \mathsf{s}_0 \oplus \{v \mapsto \mathsf{s}_0[\![e]\!], hist.v \mapsto \mathsf{s}_0[\![hist.v]\!] \frown \mathsf{s}_0[\![v]\!]\}\\
&[\![v :\in E]\!].\mathfrak{M}_0 = (\cup\{\mathsf{s} : \mathfrak{M}_0 \bullet [\![v :\in E]\!].\mathsf{s}_0\}, \mathfrak{M}_0[\![vis]\!])
\end{aligned}$$

EXAMPLE 4.4 *Suppose* $\mathcal{A} = \{a, b\}$

$$[\![\mathsf{vis}_a\ w \bullet w := 2w]\!].(\{\}, \{\})$$

$=$

$$\left(\left\{\mathsf{n} : \mathsf{D} \bullet \begin{pmatrix} hist.w \mapsto n \\ w \mapsto 2n \end{pmatrix}\right\}, w \mapsto \{a\}\right)$$

$=$ *Using the brackets ⟨⟩ for non-distinguishability*

$$\left\{\mathsf{n} : \mathsf{D} \bullet \begin{pmatrix} hist.w \mapsto \langle n \rangle_b \\ w \mapsto \langle 2n \rangle_b \end{pmatrix}\right\}$$

Sequential composition

$$[\![P \mathbin{;} Q]\!].\mathfrak{M}_0 = [\![Q]\!].[\![P]\!].\mathfrak{M}_0$$

Nondeterministic choice

The standard semantics for nondeterministic choice between two programs P and Q is as follows

$$[\![P \sqcap Q]\!].\mathfrak{M}_0 = [\![P]\!].\mathfrak{M}_0 \uplus [\![Q]\!].\mathfrak{M}_0.$$

The right hand side makes use of a disjoint union $\uplus$. A disjoint union between standard Kripke models is known to preserve modal formulas; see e.g., [8]. Classically, a disjoint union of two models is straightforward if their sets of worlds are disjoint. If that is not the case isomorphic copies of the models with disjoint sets of worlds are taken.

However, for concrete models, having only disjoint sets of worlds does not guarantee that the worlds from the two original models are not related. In addition, we have to first determine when the union of such models is well defined.

Definition 4.3 (Union of concrete models) Consider two concrete models $\mathfrak{M}_0$ and $\mathfrak{M}_1$. If $\mathfrak{M}_0$ and $\mathfrak{M}_1$ have a common set $\mathcal{P}$ of attributes then their union is well defined if *vis* agrees on $\mathcal{P}$.

The predicate *vis* agrees on $\mathcal{P}$ means that if v is a global variable in $\mathcal{P}$. In both models $\mathfrak{M}_0$ and $\mathfrak{M}_1$, the set of agents that can see v is the same.
In our setting the common set $\mathcal{P}$ consist of the global variables, and we assume that the visibility of the global variables is fixed. So *vis* agrees on $\mathcal{P}$, but not necessarily on the local variables. To take the union of two concrete models from program semantics, local variables –which corresponds to historical attribute in the semantics– must be renamed such that the two models have disjoint historical attributes.

As we noted earlier, having disjoint sets of worlds does not guarantee to construct a disjoint union of two concrete models. Accessibility relations can possibly arise between worlds from the two original models because these relations are coded in the worlds. To define disjoint union we always need to "tag" every world with the model where it comes from. This is equivalent to what is done for standard Kripke models (e.g., [8]) where isomorphic copies of the models are taken. For concrete models in particular, we can make explicit how to take the copies by virtue of Lemma 3.2.

DEFINITION 4.4 (Disjoint union of concrete models) The disjoint union $\uplus$ of two epistemic states $\mathfrak{M}_0$ and $\mathfrak{M}_1$ is defined by taking the union of their disjoint isomorphic copies constructed as follows. For each model add a new attribute which take a same value in all the world of the model, this attribute will differentiate any world from that model to any other world from the other model in the union.

The following is a classical result in modal logic, where standard Kripke models are used (see e.g., [8]). It states that modal formulas are preserved under disjoint union of models. In fact, they are preserved in both ways because the original models are generated submodels of the combined model (see also e.g., [8]).

LEMMA 4.1

$$(\mathfrak{M}_0 \uplus \mathfrak{M}_1) \models \phi \quad \equiv \quad \mathfrak{M}_0 \models \phi \wedge \mathfrak{M}_1 \models \phi$$

Atomic block

The program $«P»_A$ is executed by not allowing agents in A to see the intermediate steps in P. In $[\![«P»_A]\!].\mathfrak{M}_0$, histories in $\mathfrak{M}_0$ are preserved, histories introduced by P are made invisible from A.

Let $[\![P]\!].\mathfrak{M}_0 = \mathfrak{M}_i = (\mathsf{S}_i, \mathfrak{M}_i[\![vis]\!])$

$$[\![«P»_A]\!].\mathfrak{M}_0 = (\mathsf{S}_{\mathfrak{M}_i}, \mathfrak{M}_i[\![vis]\!] \oplus \{h : (\mathcal{H}_i \setminus \mathcal{H}_0) \bullet h \mapsto (\mathfrak{M}_i[\![vis]\!] \setminus A)\})$$

Abort

We define $\mathfrak{M}_\perp$ to be an epistemic state where any program variable can have any arbitrary value and where any agent can know the value of any program variable.

$$[\![\mathsf{abort}]\!].\mathfrak{M}_0 = \mathfrak{M}_\perp$$

Revelation

In the command $[\![\mathsf{ann!}\,\psi]\!]$ restrict the possible states to those that satisfy ψ.

$$[\![\mathsf{ann!}\,\psi]\!].\mathfrak{M}_0 = (\{\mathsf{s} : \mathfrak{M}_0 \mid (\mathfrak{M}_0, \mathsf{s}) \models \psi\}, \mathfrak{M}_0[\![vis]\!])$$

Publication

In the command $[\![\mathsf{rev}_a\{e\}]\!]$ the value of e is given to a historical variable visible to a

$$[\![\mathsf{rev}_a\{e\}]\!].\mathfrak{M}_0 = (\{\mathsf{s} : \mathfrak{M}_0 \bullet \mathsf{s} \oplus \{hist.e \mapsto \mathsf{s}[\![e]\!]\}\}, \mathfrak{M}_0[\![vis]\!] \oplus hist.e \mapsto a).$$

Magic

Magic restricts to the states that satisfy false, resulting in an empty set of possible states.

$$[\![\mathsf{magic}]\!].\mathfrak{M}_0 = (\{\}, \mathfrak{M}_0[\![vis]\!])$$

Semantics for refinement

Refinement is defined between two programs P and Q on the same set of global variables $\mathcal{P}$ and is defined for each initial configuration (epistemic state $\mathfrak{M}_0$).

$$[\![P \sqsubseteq Q]\!].\mathfrak{M}_0 \quad \equiv \quad [\![P]\!].\mathfrak{M}_0 \sqsubseteq [\![Q]\!].\mathfrak{M}_0 \tag{4.1.1}$$

The refinement in the right hand side makes use of the refinement between epistemic models given in Definition 4.6.

Refinement between models

Consider $\mathcal{V} = \mathcal{I}_f \cup \mathcal{I}_r \cup \mathcal{F} \cup \mathcal{R}$, $\mathcal{V}' = \mathcal{I}'_f \cup \mathcal{I}_r \cup \mathcal{F} \cup \mathcal{R}$, and a common set of state attributes $\mathcal{P} \subseteq \mathcal{I}_f \cap \mathcal{I}'_f$. Let us denote by $\underline{v}$ the tuple of the symbols in $\mathcal{P}$.

Definition 4.5 For a vocabulary $\mathcal{V}$ containing a set of $\mathcal{P}$ attributes. A $\mathcal{P}$-*ignorant formula* is a formula of $\mathcal{V}_{\mathcal{P}}$ such that any occurrence of a modality K_a are of the form $\neg\mathrm{K}_a\phi$, where ϕ is a classical formula. It is defined inductively on the modality on the vocabulary $\mathcal{V}_{\mathcal{P}}$ below.

$$\begin{array}{lll} \phi ::= & R(t_0, t_1, \ldots, t_{N-1}) & \theta ::= \phi \\ & \mid \neg\phi & \mid \neg \mathrm{K}_a\, \phi \\ & \mid \neg\phi & \mid \theta \wedge \theta \\ & \mid \phi \circ \phi & \mid \theta \vee \theta \\ & \mid \forall\, x \bullet \phi & \mid \forall\, x \bullet \theta \\ & \mid \exists\, x \bullet \phi & \mid \exists\, x \bullet \theta \\ & \mid (\lambda X \bullet \Phi).t & \end{array}$$

in which θ denotes a $\mathcal{P}$-ignorant formula and ϕ denotes a classical formula.

Definition 4.6 (Refinement between models) Let $\mathcal{V} = (\mathcal{I}_f, \mathcal{I}_r, \mathcal{F}, \mathcal{R})$, $\mathcal{V}' = (\mathcal{I}_f', \mathcal{I}_r, \mathcal{F}, \mathcal{R})$, and a common set of state attributes $\mathcal{P} \subseteq \mathcal{I}_f \cap \mathcal{I}_f'$. We define the *refinement* relation $\sqsubseteq_{\mathcal{P}}$ between $\mathcal{V}$ and $\mathcal{V}'$ models $\mathfrak{M}$ and $\mathfrak{M}'$ by

$$\mathfrak{M} \sqsubseteq_{\mathcal{P}} \mathfrak{M}' \quad \widehat{=} \quad \text{for any } \mathcal{P}\text{-ignorant sentence } \theta \text{ if } \mathfrak{M} \models \theta \text{ then } \mathfrak{M}' \models \theta.$$

The definition of $\mathfrak{M} \sqsubseteq_{\mathcal{P}} \mathfrak{M}'$ states that if a $\mathcal{P}$-ignorant sentence is validated in $\mathfrak{M}$ then it is validated in $\mathfrak{M}'$. Because for modal sentences validity and satisfiability are dual, any $\mathcal{P}$-ignorant sentence, that cannot be satisfied in $\mathfrak{M}'$, cannot be satisfied in $\mathfrak{M}'$. In particular we have for classical sentences:

Proposition 4.1 *Given two models $\mathfrak{M}$ and $\mathfrak{M}'$ on two vocabularies $\mathcal{V} = \mathcal{I}_f \cup \mathcal{I}_r \cup \mathcal{F} \cup \mathcal{R}$ and $\mathcal{V}' = \mathcal{I}_f' \cup \mathcal{I}_r \cup \mathcal{F} \cup \mathcal{R}$, and a common set of state attributes $\mathcal{P} \subseteq \mathcal{I}_f \cap \mathcal{I}_f'$. If φ is a classical sentence on $\mathcal{V}_{\mathcal{P}}$*

$$\textit{If } \ \mathfrak{M} \sqsubseteq_{\mathcal{P}} \mathfrak{M}' \ \textit{ then } \ \textit{if } \varphi \textit{ is satisfiable in } \mathfrak{M}' \textit{ then } \varphi \textit{ is satisfiable in } \mathfrak{M}$$

We would like to give a characterisation of $\sqsubseteq$ using the components of each model, i.e., characterising $\sqsubseteq$ semantically. To achieve that, we need the following definitions from Aumann [1] [1].

Definition 4.7 (Information sets, information partition) Consider a $\mathcal{V}-$model $\mathfrak{M} = \langle \mathsf{S}, (\sim_a)_{a \in \mathcal{A}}, \mathsf{D}_o, \mathsf{D}_c, \mathfrak{M}[\![\,.\,]\!] \rangle$ and an agent a. The *information function* I_a

[1] [1] uses Ω and ω for our S and s, $\mathbf{I}$ and $\mathcal{I}$ for our I and $\mathfrak{S}$.

associates each possible state s to its $\sim_a$equivalence class i.e., to the sets of possible states that a cannot distinguish from s.

$$\mathsf{I}_a.\mathsf{s} = \{\mathsf{s}' : \mathsf{S} \mid \mathsf{s}' \sim_a \mathsf{s}\}$$

A set $\mathsf{I}_a.\mathsf{s}$ is called an *information set* of a. The states of $\mathsf{I}_a.\mathsf{s}$ are indistinguishable from a. For two states s, s', we have either $\mathsf{I}_a.\mathsf{s} = \mathsf{I}_a.\mathsf{s}'$ or $\mathsf{I}_a.\mathsf{s} \cap \mathsf{I}_a.\mathsf{s}' = \{\}$. Thus, the information sets of a constitute a partition $\mathfrak{S}_a$ of S, which is called the *information partition* of a.

$$\mathfrak{S}_a \;=\; \mathsf{S}/\sim_a = \{\mathsf{I} \subseteq \mathsf{S} \mid \forall\, \mathsf{s}, \mathsf{t} \in \mathsf{I} \bullet \mathsf{s} \sim_a \mathsf{t}\} \;=\; \{\mathsf{s} : \mathsf{S} \bullet \mathsf{I}_a.\mathsf{s}\}$$

Recall that the possible states are partial functions from attributes to the object domain. Thus information sets are sets of partial functions. To each program variable v, $\mathsf{I}_a.\mathsf{s}.v$ is the range of values that v takes in the information set $\mathsf{I}_a.\mathsf{s}$. We will refer to $\mathsf{I}_a.\mathsf{s}.v$ as the *a-shadow of v* at s. The following defines the *a-Shadows* of v.

$$\mathfrak{S}_a.v \;\widehat{=}\; \{\mathsf{I}_a.\mathsf{s} : \mathfrak{S}_a \bullet \mathsf{I}_a.\mathsf{s}.v\} \tag{4.1.2}$$

Remark 4.1 Our use of the term "shadow" is no purpose. The set $\mathfrak{S}_a.v$ is the same as the shadow of a hidden variable in the Shadow semantics of Morgan [28] if a single agent is assumed.

Theorem 4.1 (Semantics of refinement) *Consider two $\mathcal{V}-$models $\mathfrak{M}$ and $\mathfrak{M}'$ and assume a tuple $\underline{v}$ of global variables*

$$\mathfrak{M} \sqsubseteq_{\mathcal{P}} \mathfrak{M}'$$

iff

$$\mathsf{S}'.\underline{v} \subseteq \mathsf{S}.\underline{v} \qquad \text{(Functional requirement)}$$

$$\forall\, a \in \mathcal{A} \bullet \mathfrak{S}_a.\underline{v} \subseteq_S \mathfrak{S}'_a.\underline{v} \qquad \text{(Confidentiality requirement)}$$

where $\subseteq_S$ is the Smyth partial ordering on the set of sets

$$\mathfrak{S}_a.\underline{v} \subseteq_S \mathfrak{S}'_a.\underline{v} \;\; \textit{iff} \;\; \forall\, \mathsf{I}'_a.\underline{v} \in \mathfrak{S}'_a.\underline{v} \bullet \exists\, \mathsf{I}_a.\underline{v} \in \mathfrak{S}_a.\underline{v} \mid \mathsf{I}_a.\underline{v} \subseteq \mathsf{I}'_a.\underline{v}$$

Proof of Definition 4.6 ⇒ Theorem 4.1.

Hypothesis: $\mathfrak{M} \sqsubseteq_{\mathcal{P}} \mathfrak{M}'$

$\equiv$ *Definition 4.6*

for any $\mathcal{P}$-ignorant sentence θ if $\mathfrak{M} \models \theta$ then $\mathfrak{M}' \models \theta$

Assume for contradiction $\exists\, \mathsf{I}_a.\underline{v} \mid \forall\, \mathsf{I}_a.\underline{v} \bullet \mathsf{I}_a.\underline{v} \not\subseteq \mathsf{I}_a.\underline{v}$

$\Rrightarrow$ *if* $\mathsf{t}' \sim_a \mathsf{s}'$ *then* $\mathfrak{M}[\![\underline{v}]\!].\mathsf{t}' \in \mathsf{I}_a.\underline{v}$

$\exists\, \mathsf{I}_a$ and $\mathsf{s}' \mid \forall\, \mathsf{I}_a.\underline{v} \bullet \mathsf{I}_a.\underline{v} \not\subseteq \mathsf{I}_a.\underline{v}$ and
$\mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$

$\Rrightarrow$ $\mathsf{I}_a.\underline{v} \not\subseteq \mathsf{I}_a.\underline{v}$
Truth of K_a

$\exists\, \mathsf{I}_a$ and $\mathsf{s}' \mid \mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$ and
$\forall \mathsf{s} \in \mathsf{S} \bullet \mathfrak{M}, \mathsf{s} \not\models \mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$

$\Rrightarrow$ *Truth of negation*

$\exists\, \mathsf{I}_a$ and $\mathsf{s}' \mid \mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$ and
$\forall \mathsf{s} \in \mathsf{S} \bullet \mathfrak{M}, \mathsf{s} \models \neg\mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$

$\Rrightarrow$ *Definition of validity*

$\exists\, \mathsf{I}_a$ and $\mathsf{s}' \mid \mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$ and
$\mathfrak{M} \models \neg\mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$

$\Rrightarrow$ $\neg\mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$ *is ignorant*
Hypothesis

$\exists\, \mathsf{I}_a$ and $\mathsf{s}' \mid \mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$ and
$\mathfrak{M}' \models \neg\mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$

$\Rrightarrow$ *Definition of validity*

$\mathfrak{M}' \not\models \neg\mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$ and $\mathfrak{M}' \models \neg\mathrm{K}_a(\underline{v} \in \mathsf{I}_a.\underline{v})$

$\Rrightarrow$

False

$\square$

LEMMA 4.2 *Consider $\mathcal{P} = \{v_0, \ldots, v_N\}$ and denote $\underline{v} = (v_0, \ldots, v_{N-1})$. If ϕ is a $\mathcal{V}_{\mathcal{P}}$ non-modal sentence and $\underline{n}$ is an N-uple of rigid concepts, we have*

$$\underline{v} \doteq \underline{n} \Rightarrow \phi \textit{ is satisfiable} \quad \equiv \quad \underline{v} \doteq \underline{n} \Rightarrow \phi \textit{ is valid} \tag{4.1.3}$$

Proof. Because ϕ is non-modal, $\mathfrak{M}, \mathsf{s} \models \underline{v} \doteq \underline{n} \Rightarrow \phi$ iff $\mathfrak{M}, \mathsf{s} \models \phi_{\underline{v}\backslash\underline{n}}$. Because all flexible concepts are in $\underline{v}$ all terms in $\phi_{\underline{v}\backslash\underline{n}}$ are rigid, thus $\phi_{\underline{v}\backslash\underline{n}}$ is valid. Therefore $\underline{v} \doteq \underline{n} \Rightarrow \phi$ is valid. □

Proof of Theorem 4.1 $\Rrightarrow$ *Definition 4.6.* The proof is by structural induction on the formula θ. We take as base case a classical formula ϕ (without modality) and prove the implication for an ignorant formula $\neg\mathrm{K}\phi$. The rest of the proof treats the compositions of the first two kinds with usual connectives $\vee, \wedge, \exists, \forall$ and are not presented here.

Assume that $\forall\, a \in \mathcal{A} \bullet \mathfrak{S}_a.\underline{v} \subseteq_S \mathfrak{S}'_a.\underline{v}$.

Case $\theta = \phi$

Let us prove the contraposition $\mathfrak{M}' \not\models \phi \Rrightarrow \mathfrak{M} \not\models \phi$.

The partial ordering $\mathfrak{S}_a.\underline{v} \subseteq_S \mathfrak{S}'_a.\underline{v}$ implies that $\bigcup \mathfrak{S}'_a.\underline{v} \subseteq \bigcup \mathfrak{S}_a.\underline{v}$. Thus the range of values of $\underline{v}$ in $\mathfrak{M}'$ is contained in its range of values in $\mathfrak{M}$. Therefore, if a non-modal $\underline{v}$-sentence ϕ is satisfiable in $\mathfrak{M}'$, then ϕ is satisfiable in $\mathfrak{M}$.

Now assume that $\mathfrak{M}' \not\models \phi$. This implies that $\neg\phi$ is satisfiable in $\mathfrak{M}$. By the previous observation $\neg\phi$ is also satisfiable in $\mathfrak{M}$ thus $\mathfrak{M} \not\models \phi$.

Case $\theta = \neg\mathrm{K}_a\phi$ We will prove the contraposition $\mathfrak{M}' \not\models \neg\mathrm{K}_a\phi \Rrightarrow \mathfrak{M} \not\models \phi$.

$\mathfrak{M}' \not\models \neg\mathrm{K}_a\phi$

$\Rrightarrow$ *Duality of validity and satisfiability*

There exists s' such that $\mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a\phi$

$\Rrightarrow$ I_a *is* $\sim_a$*-equivalence class of* s'

There exists s' and I_a such that $\mathfrak{M}', \mathsf{s}' \models \mathrm{K}_a\phi$

$\Rrightarrow$ *Definition of* K_a

There exists I_a such that $\forall\, \mathsf{s}' \in \mathsf{I}_a \bullet \mathfrak{M}', \mathsf{s}' \models \phi$

$\Rrightarrow$

For each $\mathsf{s}'_i \in \mathsf{I}_a$, let n be the rigid concept associated to $\mathsf{s}'.v$;
then $\mathfrak{M}', \mathsf{s}' \models \underline{v} = \underline{n} \Rightarrow \phi$

$\Rrightarrow$ *Satisfiability*

For concept n associated to a $\mathsf{v} \in \mathsf{I}_a.\underline{v}$, $v \underline{=n} \Rightarrow \phi$ is satisfiable in $\mathfrak{M}'$

$\Rrightarrow$ *The claim is true for $v = n \Rightarrow \phi$ which is non-modal*

For each concept n associated to a $\mathsf{v} \in \mathsf{I}_a.\underline{v}$, $v = n \Rightarrow \phi$ is satisfiable in $\mathfrak{M}$

$\Rrightarrow$ *Lemma 4.2*

For each concept n associated to a $\mathsf{v} \in \mathsf{I}_a.\underline{v}$, $\underline{v} = \underline{n} \Rightarrow \phi$ is valid in $\mathfrak{M}$ (*)

$\Rrightarrow$ *Hypothesis*

For each concept n associated to a $\mathsf{v} \in \mathsf{I}_a.\underline{v}$, $\underline{v} = \underline{n} \Rightarrow \phi$ is valid in $\mathfrak{M}$ and there exists I_a such that $\mathsf{I}_a.\underline{v} \subseteq \mathsf{I}_a.\underline{v}$

$\Rrightarrow$ $\mathsf{s} \in \mathsf{I}_a \Rrightarrow \mathsf{s}.\underline{v} \in \mathsf{I}_a.v \subseteq \mathsf{I}_a.\underline{v}$

For each concept n associated to a $\mathsf{v} \in \mathsf{I}_a.\underline{v}$, $v = n \Rightarrow \phi$ is valid in $\mathfrak{M}$ and there exists I_a such that for each $\mathsf{s} \in \mathsf{I}_a$, $\mathfrak{M}, \mathsf{s} \models \underline{v} = \underline{n}$ for some $\underline{n} \in \mathsf{I}_a$

$\Rrightarrow$ *From (*), truth of $\Rightarrow$*

There exists I_a such that for each $\mathsf{s} \in \mathsf{I}_a$, $\mathfrak{M}, \mathsf{s} \models \phi$

$\Rrightarrow$ *Truth of K_a*
I_a is a $\sim_a$ class

There exists I_a such that for each $\mathsf{s} \in \mathsf{I}_a$, $\mathfrak{M}, \mathsf{s} \models \mathrm{K}_a\phi$

$\Rrightarrow$ *Satisfiability*

$\mathrm{K}_a\phi$ is satisfiable in $\mathfrak{M}$

$\Rrightarrow$ *Validity*

$\mathfrak{M} \not\models \neg\mathrm{K}_a\phi$

$\square$

We note that the functional requirement in Theorem 4.1 is the same requirement as for the refinement of classical programs 1.2.2. This correspond to the property that refinement preserves valid factual formulas.

$$\llbracket \mathsf{skip} \rrbracket.\mathfrak{M}_0 = \mathfrak{M}_0$$

$$\llbracket P \fatsemi Q \rrbracket.\mathfrak{M}_0 = \llbracket Q \rrbracket.\llbracket P \rrbracket.\mathfrak{M}_0$$

$$\llbracket P \sqcap Q \rrbracket.\mathfrak{M}_0 = \llbracket P \rrbracket.\mathfrak{M}_0 \uplus \llbracket Q \rrbracket.\mathfrak{M}_0$$

$$\begin{aligned}
\llbracket \mathsf{vis}_a\, w \rrbracket.\mathsf{s}_0 &= \{n : \mathsf{D} \bullet \mathsf{s}_0 \oplus \{w \mapsto n\}\} \\
\llbracket \mathsf{vis}_a\, w \rrbracket.\mathfrak{M}_0 &= \left(\bigcup \{\mathsf{s}_0 : \mathsf{S}_0 \bullet \llbracket \mathsf{vis}_a\, w \rrbracket.\mathsf{s}_0\}\ ,\ \mathfrak{M}_0\llbracket vis \rrbracket \oplus \{w \mapsto \{a\}\}\right)
\end{aligned}$$

$$\begin{aligned}
\llbracket w\, \mathsf{end} \rrbracket.\mathsf{s}_0 &= (\{w\} \lhd \mathsf{s}_0) \oplus \{hist.w \mapsto \mathsf{s}_0\llbracket hist.w \rrbracket \frown \mathsf{s}_0\llbracket w \rrbracket\}) \\
\llbracket w\, \mathsf{end} \rrbracket.\mathfrak{M}_0 &= (\{\mathsf{s}_0 : \mathsf{S}_0 \bullet \llbracket w\, \mathsf{end} \rrbracket.\mathsf{s}_0\}, \{w\} \lhd \mathfrak{M}_0\llbracket vis \rrbracket)
\end{aligned}$$

$$\begin{aligned}
\llbracket v := e \rrbracket.\mathsf{s}_0 &= \{\mathsf{s}_0 \oplus \{v \mapsto \mathsf{s}_0\llbracket e \rrbracket, hist.v \mapsto \mathsf{s}_0\llbracket hist.v \rrbracket \frown \mathsf{s}_0\llbracket v \rrbracket\}\} \\
\llbracket v := e \rrbracket.\mathfrak{M}_0 &= (\{\mathsf{s} : \mathfrak{M}_0 \bullet \llbracket v := e \rrbracket.\mathsf{s}_0\}, \mathfrak{M}_0\llbracket vis \rrbracket)
\end{aligned}$$

$$\begin{aligned}
\llbracket v :\in E \rrbracket.\mathsf{s}_0 &= \{e : E \bullet \mathsf{s}_0 \oplus \{v \mapsto \mathsf{s}_0\llbracket e \rrbracket, hist.v \mapsto \mathsf{s}_0\llbracket hist.v \rrbracket \frown \mathsf{s}_0\llbracket v \rrbracket\}\} \\
\llbracket v :\in E \rrbracket.\mathfrak{M}_0 &= \left(\bigcup \{\mathsf{s} : \mathfrak{M}_0 \bullet \llbracket v :\in E \rrbracket.\mathsf{s}_0\}, \mathfrak{M}_0\llbracket vis \rrbracket\right)
\end{aligned}$$

$$\llbracket \text{«}P\text{»}_A \rrbracket.\mathfrak{M}_0 = (\mathsf{S}_{\mathfrak{M}_i}, \mathfrak{M}_i\llbracket vis \rrbracket \oplus \{h : (\mathcal{H}_i \setminus \mathcal{H}_0) \bullet h \mapsto (\mathfrak{M}_i\llbracket vis \rrbracket \setminus A)\}))$$

$$\llbracket \mathsf{ann!}\, \phi \rrbracket.\mathfrak{M}_0 = (\{\mathsf{s} : \mathfrak{M}_0 \mid (\mathfrak{M}_0, \mathsf{s}) \models p\}, \mathfrak{M}_0\llbracket vis \rrbracket) \quad (= \mathfrak{M}_{0|\phi})$$

$$\llbracket \mathsf{rev}_a\{e\} \rrbracket.\mathfrak{M}_0 = (\{\mathsf{s} : \mathfrak{M}_0 \bullet \mathsf{s} \oplus \{hist.e \mapsto \mathsf{s}\llbracket e \rrbracket\}\}, \mathfrak{M}_0\llbracket vis \rrbracket \oplus hist.e \mapsto a)$$

$$\llbracket \mathsf{magic} \rrbracket.\mathfrak{M}_0 = (\{\}, \mathfrak{M}_0\llbracket vis \rrbracket)$$

$$\llbracket \mathsf{abort} \rrbracket.\mathfrak{M}_0 = \mathfrak{M}_\perp$$

$$\llbracket \mathsf{assert}\, p \rrbracket.\mathfrak{M}_0 = (\mathsf{S}_{0|p}, \mathfrak{M}_0\llbracket vis \rrbracket) \uplus \mathfrak{M}_\perp$$

FIGURE 4.1: Denotational semantics of programs.

4.2 Weakest precondition semantics

In this section, we extend the weakest precondition given in [28].

4.2.1 WP for reveal

The formula $\mathrm{wp}[\![\mathsf{ann!}\, p]\!].\phi$ is the precondition such that after revealing p, ϕ is true and corresponds directly to the public announcement formula $[p]\phi$. Recall from Section 3.3, that a formula $[p]\phi$ reads as after every truthful public announcement of p, ϕ true.

Proposition 4.2 *The weakest precondition semantics are as follows for programs excluding explicit atomicity*

$$\begin{aligned}
\mathrm{wp}[\![\mathsf{skip}]\!].\phi &= \phi && (4.2.1)\\
\mathrm{wp}[\![\mathsf{assert}\, p]\!].\phi &= \langle p\rangle\phi && (4.2.2)\\
\mathrm{wp}[\![\mathsf{ann!}\, p]\!].\phi &= [p]\phi && (4.2.3)\\
\mathrm{wp}[\![\mathsf{rev}_a\ e]\!].\phi &= \forall\, x \in \mathit{vis}.a \bullet [x \doteq e]\phi && (4.2.4)\\
\mathrm{wp}[\![v := e]\!].\phi &= \forall\, x \overset{\mathit{vis}}{\sim} v \bullet [x \doteq e]\phi_{v\backslash x} && (4.2.5)\\
\mathrm{wp}[\![v :\in E]\!].\phi &= \forall\, x \overset{\mathit{vis}}{\sim} v \bullet [x \in E]\phi_{v\backslash x} && (4.2.6)\\
\mathrm{wp}[\![P \fatsemi Q]\!].\phi &= \mathrm{wp}[\![P]\!].(\mathrm{wp}[\![Q]\!].\phi) && (4.2.7)\\
\mathrm{wp}[\![P \sqcap Q]\!].\phi &= \mathrm{wp}[\![P]\!].\phi \wedge \mathrm{wp}[\![Q]\!].\phi && (4.2.8)\\
\mathrm{wp}[\![P \lhd b \rhd Q]\!].\phi &= [b]\mathrm{wp}[\![P]\!].\phi \wedge [\neg b]\mathrm{wp}[\![Q]\!].\phi && (4.2.9)\\
\mathrm{wp}[\![[\![\, \mathsf{vis}_a\ v \bullet P]\!]]\!].\phi &= \forall\, v \in \mathit{vis}.a \bullet \mathrm{wp}[\![P]\!].\phi && (4.2.10)
\end{aligned}$$

where $x \overset{\mathit{vis}}{\sim} y \ \equiv\ (x \in \mathit{vis}.a) \Leftrightarrow (y \in \mathit{vis}.a)$.

Example 4.5 (Atomic block in wp) *Consider two distinct agents a and b.*

$$\begin{aligned}
&\mathrm{wp}[\![\langle\!\langle\, \mathsf{ann!}\, p \sqcap \mathsf{skip}\,\rangle\!\rangle_b]\!].(\neg \mathrm{K}_b \neg \mathrm{K}_a p)\\
= \quad & \qquad\qquad \text{wp } \textit{for } \overline{a}\\
&\neg \mathrm{K}_b \mathrm{wp}[\![(\mathsf{ann!}\, p \sqcap \mathsf{skip})]\!].\neg \mathrm{K}_a p\\
= \quad & \qquad\qquad \text{wp } \textit{for a on } \sqcap\\
&\neg \mathrm{K}_b\left((\mathrm{wp}[\![\mathsf{ann!}\, p]\!].\neg \mathrm{K}_a p) \wedge (\mathrm{wp}[\![\mathsf{skip}]\!].\neg \mathrm{K}_a p)\right)\\
= \quad & \qquad\qquad \text{wp } \textit{for on } \mathsf{ann!}\ \textit{and } \mathsf{skip}\\
&\neg \mathrm{K}_b\left([p]\neg \mathrm{K}_a p \wedge \neg \mathrm{K}_a p\right)\\
= \quad & \qquad\qquad \textit{public announcement logic}\\
&\neg \mathrm{K}_b\left(\mathsf{false} \wedge \neg \mathrm{K}_a p\right)
\end{aligned}$$

$=$

true

EXAMPLE 4.6 *This example illustrates the difference between an atomic choice and an explicit nondeterministic choice using the wp-semantics.*

$$\begin{aligned}
\mathrm{wp}[\![v :\in \{0,1\}]\!].\phi &= \forall\, x \overset{vis}{\sim} v \bullet [x \in \{0,1\}]\phi_{v\backslash x} \\
\mathrm{wp}[\![v := 0 \sqcap v := 1]\!].\phi &= \left(\wedge \begin{array}{l} \forall\, x \overset{vis}{\sim} v \bullet [x = 0]\phi_{v\backslash x} \\ \forall\, x \overset{vis}{\sim} v \bullet [x = 1]\phi_{v\backslash x} \end{array} \right) \\
&= \forall\, x \overset{vis}{\sim} v \bullet \left(\wedge \begin{array}{l} [x = 0]\phi_{v\backslash x} \\ [x = 1]\phi_{v\backslash x} \end{array} \right)
\end{aligned}$$

4.2.2 WP for explicit atomicity

Recall that the program $\langle\!\langle P \rangle\!\rangle_A$ allows only agents not in A to observe the steps within P. The formula $\mathrm{wp}[\![\langle\!\langle P \rangle\!\rangle_A]\!].\phi$ is defined inductively on ϕ.

PROPOSITION 4.3 *The following are the wp-semantics for explicit atomicity, in which A is a group of agents containing a but not b.*

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\alpha = \mathrm{wp}[\![P]\!].\alpha \tag{4.2.11}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\neg\phi = \neg\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi \tag{4.2.12}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_0 \wedge \phi_1 = \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_0 \wedge \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_1 \tag{4.2.13}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_0 \vee \phi_1 = \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_0 \vee \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_1 \tag{4.2.14}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].(\forall\, x \bullet \phi) = \forall\, x \bullet \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi \tag{4.2.15}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].(\exists\, x \bullet \phi) = \exists\, x \bullet \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi \tag{4.2.16}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_0 \rightarrow \phi_1 = \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_0 \rightarrow \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi_1 \tag{4.2.17}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\mathrm{K}_b\phi = \mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi \tag{4.2.18}$$

$$\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\mathrm{K}_a\phi = \mathrm{K}_a(\mathrm{wp}[\![\text{«}P\text{»}_A]\!].\phi) \tag{4.2.19}$$

WP for refinement

With the same definition of ignorant formula as in Chapter 3, we give the wp-semantics for refinement below.

$$P \sqsubseteq Q \quad \equiv \quad \text{For any ignorant formula } \theta \;\; \mathrm{wp}[\![P]\!].\theta \Rightarrow \mathrm{wp}[\![Q]\!].\theta$$

Proposition 4.4

$$P \sqsubseteq Q \quad \equiv \quad \textit{For any classical formula } \phi \quad \left(\begin{array}{l} \mathrm{wp}[\![P]\!].\phi \Rightarrow \mathrm{wp}[\![Q]\!].\phi \\ \wedge \\ \mathrm{wp}[\![P]\!].\neg \mathrm{K}\phi \Rightarrow \mathrm{wp}[\![Q]\!].\neg \mathrm{K}\phi \end{array} \right)$$

Proof. Other ignorant formulas are conjunctions, disjunctions and quantifications on $\neg$K. Our definition of wp$[\![P]\!]$ (standard program) and wp$[\![«P»]\!]$ gives predicate transformers monotone with respect to $\wedge, \vee, \exists, \forall$. For the case of standard programs (wp$[\![P]\!]$), the reason is that both $[]$ and $\langle\rangle$ are monotone with respect to $\wedge, \vee, \exists, \forall$ (Law 3.3.14, 3.3.15, 3.3.18, and 3.3.17). For atomic programs, wp$[\![«P»]\!]$ is monotone with respect to $\wedge, \vee, \exists, \forall$ by the Laws 4.2.13, 4.2.14, 4.2.16, and 4.2.15. □

4.3 Connection between the two semantics

Instead of the classical denotational semantics relating an initial state to a set of final states, the semantics given in Section 4.1 relates an initial set of states to a final set of states. This is equivalent to a strongest postcondition semantics. Thus, for a given command P the connection between the two semantics of P is given by the equivalence

$$\mathfrak{M}_0 \models \mathrm{wp}[\![P]\!].\psi \quad \equiv \quad [\![P]\!].\mathfrak{M}_0 \models \psi \tag{4.3.1}$$

Connection for public announcement

$$\mathfrak{M}_0 \models \mathrm{wp}[\![\mathsf{ann!}\,\phi]\!].\psi$$

$\equiv$

$$\mathfrak{M}_0 \models [\phi]\psi$$

$\equiv$

$$\mathfrak{M}_{0|\phi} \models \psi$$

$\equiv$

$[\![\mathsf{ann!}\,\phi]\!].\mathfrak{M}_0 \models \psi$

Connection for sequential composition

Assume for induction that the equivalence is true for commands P and Q.

$\mathfrak{M}_0 \models \mathrm{wp}[\![P \fatsemi Q]\!].\psi$

$\equiv$ wp$[\![]\!]$ *semantics of* $\fatsemi$

$\mathfrak{M}_0 \models \mathrm{wp}[\![P]\!].\mathrm{wp}[\![Q]\!].\psi$

$\equiv$ *Induction hypothesis*

$[\![P]\!].\mathfrak{M}_0 \models \mathrm{wp}[\![Q]\!].\psi$

$\equiv$ *Induction hypothesis*

$[\![Q]\!].[\![P]\!].\mathfrak{M}_0 \models \psi$

$\equiv$ $[\![]\!]$ *semantics of* $\fatsemi$

$[\![P \fatsemi Q]\!].\mathfrak{M}_0 \models \psi$

Connection for nondeterministic choice

A nondeterministic choice of programs corresponds to disjoint union of models. The connection makes use of the preservation of modal formulas under disjoint union (4.1).

Assume that the equivalence is true for commands P and Q.

$\mathfrak{M}_0 \models \mathrm{wp}[\![P \sqcap Q]\!].\psi$

$\equiv$ wp$[\![]\!]$ *semantics of* $\sqcap$

$\mathfrak{M}_0 \models \mathrm{wp}[\![p]\!].\psi \wedge \mathrm{wp}[\![q]\!].\psi$

$\equiv$ *truth of* $\wedge$

$\mathfrak{M}_0 \models \mathrm{wp}[\![p]\!].\psi \;\wedge\; \mathfrak{M}_0 \models \mathrm{wp}[\![q]\!].\psi$

$\equiv$ *induction hypothesis*

$[\![p]\!].\mathfrak{M}_0 \models \psi \;\wedge\; [\![q]\!].\mathfrak{M}_0 \models \psi$

$\equiv$ *Lemma 4.1*

$$(\llbracket P \rrbracket.\mathfrak{M}_0 \uplus \llbracket Q \rrbracket.\mathfrak{M}_0) \models \psi$$

4.4 Discussion

The denotational model given in this chapter gives an operational understanding of programs. But the weakest precondition semantics might be easier to handle in calculations. Both the denotational model and the weakest precondition model can be used to prove the soundness of the programming laws, which are the subject of the next chapter.

Chapter 5

Algebraic laws

In this chapter, we will give programming laws that are sensitive to information flow. These laws allow us to reason about programs using mostly the same language in which they are written, abstracting from the models described in the previous chapter. The model will be used only to prove the soundness of the proposed laws at the end of this chapter. We begin by motivating the use of the laws.

5.1 On the use of program algebra

The use of algebraic laws allows for instance a formal stepwise refinement of programs [27, 20, 2]. Using this approach, from the specification of a program, we can derive an implementation that necessarily meets the same requirements of by virtue of the laws.

Reasoning about confidentiality and deriving security protocols

Stepwise derivation of a security protocol begins by writing the specification S of the protocol using the programming language described in Chapter 2. By means of algebraic laws, S is refined until we get an implementation I. Each step of refinement should obey the programming laws.

$$S = P_0 \sqsubseteq P_1 \sqsubseteq \ldots \sqsubseteq P_{n-1} = I$$

The derived implementation I achieves the functional requirements of S. Moreover I does not reveal any more information than S does.

This type of derivation was used to develop security protocols such as the dining cryptographers protocol, an oblivious transfer protocol, and the millionaire's protocol in [28, 29, 23, 25, 26].

Describing knowledge-based programs and reasoning about information flow

Because ignorance and knowledge are dual, the algebraic laws can also be used to reason about what agents can learn. Suppose that we are given how a system works, and described it in terms of our programming language. We are given a program P with global variables in a set $\mathcal{G}$. Consider a classical formula ϕ on the global program variables. The question *Can the agents learn that ϕ holds after any run of P?* is answered positively if we have the refinement

$$P \sqsubseteq P \fatsemi \mathsf{ann!}\,\phi \tag{5.1.1}$$

If the execution of P followed by the explicit public announcement of ϕ reveals no more information than the execution of P, then P implicitly reveals ϕ.

5.2 Laws

Morgan [29] listed five principles that would ease the practical use of ignorance-preserving refinement calculus.

Refinement is monotonic. If one program fragment is shown in isolation to refine another, then the refinement continues to hold in any context.

All classical visible-variable only refinements remain valid.

All classical structural refinements remain valid.

Some classical explicit-hidden refinements become invalid.

Referential transparency. If two expressions are equal (in a declarative context) then they may be exchanged (in that context) without changing the meaning of the program.

For the first principle, we note that all operations on programs except for atomic execution are monotonic with respect to refinement; see [26]. For the principles 2,3 and 4 we refer to the arguments in [30] which we believe can be generalised to multi-agents. Our treatment of equality in the first-order epistemic logic (Chapter 3) prevents us from any problem with referential transparency.

5.2.1 Laws of refinement

Law 5.1 defines the refinement relation using nondeterminism as for classical programs. This law still holds because we assume that the nondeterministic choice $\sqcap$ is an explicit choice, thus avoiding the refinement paradox (see Section 1.3).

LAW 5.1 $P \sqsubseteq Q \quad \equiv \quad P \sqcap Q = P$

LAW 5.2 $\mathsf{ann!}\,\phi = \mathsf{ann!}\,\psi \quad \equiv \quad \phi \leftrightarrow \psi$

REMARK 5.1 It may seem natural to consider the refinement between two public announcements such as

$$\mathsf{ann!}\,x = 2 \ \sqsubseteq \ \mathsf{ann!}\,x \text{ is even.} \tag{5.2.1}$$

Announcing that x is even reveals no more information than announcing that $x = 2$. But such refinement is correct only if $x = 2$ is true. At a state where $x = 4$, the refinement does not hold because the left hand side would be miraculous (magic). However, in our reasoning we never refer to what is the actual state. And this kind of refinement is not valid. Instead, a similar refinement that is valid independently of the actual state uses the revelation command instead of the public announcement. We have

$$\mathsf{rev}\ (x = 2) \sqsubseteq \mathsf{rev}\ (x \ \mathsf{mod}\ 2) \tag{5.2.2}$$

where $x = 2$ is a boolean expression (T or F).

5.2.2 Structural laws

Following are examples of structural laws, we refer to [34] for more laws.

LAW 5.3 *For non-empty sets of programs $\mathcal{P}$ and $\mathcal{Q}$*

$$(\bigsqcap \mathcal{P}) \sqcap (\bigsqcap \mathcal{Q}) \;=\; \bigsqcap(\mathcal{P} \cup \mathcal{Q}).$$

LAW 5.4

$$P \triangleleft b \triangleright (Q \sqcap R) \;=\; (P \triangleleft b \triangleright Q) \sqcap (P \triangleleft b \triangleright R)$$

LAW 5.5

$$(P \sqcap Q) \,⨾\, R \;=\; (P \,⨾\, R) \sqcap (Q \,⨾\, R)$$

5.2.3 Laws on explicit atomicity

The atomic bracket $\langle\!\langle\rangle\!\rangle_a$ has the following properties (Morgan): at run time, a weak agent a is assumed there (an advantage); but, at development time, refinement is not allowed there (a disadvantage).

$$P \sqsubseteq Q \;\;\not\Rightarrow\;\; \langle\!\langle P \rangle\!\rangle \sqsubseteq \langle\!\langle Q \rangle\!\rangle \tag{5.2.3}$$

The Laws 5.7, 5.8, and 2.2 (Chapter 2) state that for syntactically atomic programs $\langle\!\langle P \rangle\!\rangle = P$. Syntactically atomic programs include assignment, atomic choice, and the announcement of a formula but exclude the revelation of an expression (Law 2.3).

LAW 5.6 *For any program P, $P \sqsubseteq \langle\!\langle P \rangle\!\rangle$*

LAW 5.7 $\langle\!\langle x := e \rangle\!\rangle \;=\; x := e$

LAW 5.8 $\langle\!\langle x :\in E \rangle\!\rangle \;=\; x :\in E$

5.2.4 Laws on sequential composition

In classical programs, the sequential composition of two coercions can be merged into one coercion using the conjunction of the two logical expressions. In general, this does not apply for the sequential composition of public announcements (which corresponds to coercions for classical programs, see Section 2.2.6), see Example 5.1. The sequential composition of public announcements requires the use of the diamond PAL formula (Law 5.9). Law 5.12 states that publications can commute between them.

LAW 5.9 $\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi \;=\; \mathsf{ann!}\langle\phi\rangle\psi.$

LAW 5.10 $\mathsf{ann!}\,\phi = \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi \;\equiv\; [\phi]\psi$

LAW 5.11 *For a classical formula φ and any formula ψ*

$$\mathsf{ann!}\,\psi \fatsemi \mathsf{ann!}\,\varphi = \mathsf{ann!}(\psi \wedge \varphi)$$

LAW 5.12 $\mathsf{rev}_{a_0}\{e_0\} \fatsemi \mathsf{rev}_{a_1}\{e_1\} = \mathsf{rev}_{a_0}\{e_0\} \fatsemi \mathsf{rev}_{a_1}\{e_1\}$

The following example shows that Law 5.11 does not hold for non-classical formulas.

EXAMPLE 5.1 ($\mathsf{ann!}\,\psi \fatsemi \mathsf{ann!}\,\varphi \neq \mathsf{ann!}\,(\psi \wedge \varphi)$) *Consider two agents a, b and program variables p, q. In the scope of*

$$\mathsf{hid}_{\{a,b\}}\, p : \mathbb{B} \fatsemi \mathsf{hid}_{\{b\}}\, q : \mathbb{B} \fatsemi \mathsf{ann!}\, p \vee q$$

we have

$$\mathsf{ann!}\, p \fatsemi \mathsf{ann!}\, \mathrm{K}_a p \not\sqsubseteq \mathsf{ann!}\,(p \wedge \mathrm{K}_a p) \tag{5.2.4}$$

The right hand side reveals the value of q to b. The formula $(p \wedge \mathrm{K}_a p)$ is true only at $(p, q) = (1, 0)$, which would be the only final state. But the left hand side does not reveal q to b. Announcing that p only discards the state $(p, q) = (0, 1)$. The possible final states are $(p, q) = (1, 1)$ and $(p, q) = (1, 0)$, and announcing $\mathrm{K}_a p$ afterwards will not change them.

5.2.5 Successful formulas

The equality $\mathsf{ann!}\,\psi \fatsemi \mathsf{ann!}\,\phi = \mathsf{ann!}(\psi \wedge \phi)$ is not always valid as we have seen in the previous example. Formulas satisfying this equality play an important role in reasoning about public announcement and in Dynamic Epistemic Logic (DEL). They are defined in the following.

DEFINITION 5.1 (Successful formula) A successful formula is a formula that remains true after being revealed, i.e., $[\phi]\phi$.

DEFINITION 5.2 (Preserved formula) Preserved formulas are a fragment of the logical formulas defined in BNF as follows

$$\begin{aligned}
\alpha ::= &\ R(t_0, t_1, \ldots, t_{N-1}) \\
\phi ::= &\ \alpha \\
&\mid \neg\alpha \\
&\mid \mathrm{K}_a\, \phi \\
&\mid \phi \wedge \phi \\
&\mid \phi \vee \phi \\
&\mid \forall\, x \bullet \phi \\
&\mid \exists\, x\, \phi
\end{aligned}$$

in which ϕ denotes a preserved formula and α denotes an atomic formula.

Proposition 5.1 *Preserved formulas are successful [11].*

Law 5.13 *For a successful formula ϕ:*

$$\mathsf{ann!}\,\phi \;=\; \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\phi$$

Law 5.14 *For preserved formula ϕ and any formula ψ:*

$$\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi \;=\; \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi \fatsemi \mathsf{ann!}\,\phi$$

Law 5.15 *For a successful formula ϕ, any formula ψ, $a \in \mathcal{A}$, and $A \subseteq \mathcal{A}$:*

$$\begin{aligned}
\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi &= \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}(\phi \wedge \psi) \\
\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi &= \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}(\mathrm{K}_a\phi \wedge \psi) \\
\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi &= \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}(\mathrm{C}_A\phi \wedge \psi)
\end{aligned}$$

where $a \in \mathcal{A}, A \subseteq \mathcal{A}$, and C_A designates the common knowledge modality (Section 1.4).

Law 5.15 will be used particularly in our treatment of the Three Wise Men and the Muddy Children Puzzles.

5.3 Soundness of the laws

Soundness of Law 5.2: $\mathsf{ann!}\,\phi = \mathsf{ann!}\,\psi \;\equiv\; \phi \Leftrightarrow \psi$.

$\quad \mathsf{ann!}\,\phi = \mathsf{ann!}\,\psi$

$\equiv$

$\quad \forall$ ignorant formula θ, $\mathrm{wp}.(\mathsf{ann!}\,\phi).\theta \;\Leftrightarrow\; \mathrm{wp}.(\mathsf{ann!}\,\psi).\theta$

$\equiv$

$\quad \forall$ ignorant formula θ, $[\phi]\theta \;\Leftrightarrow\; [\psi]\theta$

$\Rrightarrow$ false *is ignorant and atomic*

$\quad \phi \to \mathsf{false} \;\Leftrightarrow\; \psi \to \mathsf{false}$

$\Rrightarrow$

$\quad \phi \;\Leftrightarrow\; \psi$

$\Rrightarrow$ *Admissible rule of PAL*

$\quad$ for any χ, $[\phi]\chi \;\Leftrightarrow\; [\psi]\chi$

$\Rrightarrow$ *In particular for ignorant formulas*

$\quad \mathsf{ann!}\,\phi = \mathsf{ann!}\,\psi$

□

LEMMA 5.1 $[\phi]\phi \;\Leftrightarrow\; \phi \leftrightarrow \langle\phi\rangle\phi$

Proof.

$\quad \mathsf{true} \leftrightarrow [\phi]\phi$

$\Rightarrow$ (3.3.22)

$\quad \phi \leftrightarrow \langle\phi\rangle\phi$

$\Rightarrow$

$\quad \neg\phi \vee \phi \leftrightarrow \langle\phi\rangle\phi \vee \neg\phi$

$\Rightarrow$

$\quad \mathsf{true} \leftrightarrow \phi \to \langle\phi\rangle\phi$

$\Rightarrow$ (3.3.25)

$\quad \mathsf{true} \leftrightarrow [\phi]\phi$

□

Soundness of Law 5.9. Let θ be an ignorant formula (although the derivation is true for any arbitrary formula).

$\mathrm{wp}[\![\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi]\!].\theta$

$\Leftrightarrow$ *wp-semantics of* $\fatsemi$

$\mathrm{wp}[\![\mathsf{ann!}\,\phi]\!].\mathrm{wp}[\![\mathsf{ann!}\,\psi]\!].\theta$

$\Leftrightarrow$ *wp-semantics of* $\mathsf{ann!}$

$[\phi][\psi]\theta$

$\Leftrightarrow$ $[]-$*composition* (3.3.27)

$[\phi \wedge [\phi]\psi]\theta$

$\Leftrightarrow$ $[]-$*definition of* $\langle\rangle$ (3.3.22)

$[\langle\phi\rangle\psi]\theta$

$\Leftrightarrow$

$\mathrm{wp}[\![\mathsf{ann!}\langle\phi\rangle\psi]\!].\theta$

□

Proof of Law 5.10. The implication $\Leftarrow$ follows from the previous law as follows:

$\mathrm{wp}[\![\mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi]\!].\theta$

$\Leftrightarrow$

$\mathrm{wp}[\![\mathsf{ann!}\langle\phi\rangle\psi]\!].\theta$

$\Leftrightarrow$

$[\langle\phi\rangle\psi]\theta$

$\Leftrightarrow$ *Proposition 3.3.22*

$[\phi \wedge [\phi]\psi]\theta$

$\Leftrightarrow$ *Assumption* $[\phi]\psi$

$[\phi]\theta$

$\Leftrightarrow$

$\mathrm{wp}[\![\mathsf{ann!}\,\phi]\!].\theta$

We need to prove $\Rrightarrow$.

$\mathsf{ann!}\,\phi \;=\; \mathsf{ann!}\,\phi \fatsemi \mathsf{ann!}\,\psi$

$\equiv$

$\mathrm{wp}[\![\mathsf{ann!}\,\phi]\!].\theta \Leftrightarrow \mathrm{wp}[\![\mathsf{ann!}\,\phi \mathbin{⨟} \mathsf{ann!}\,\psi]\!].\theta$ for any ignorant formula θ

$\Rrightarrow$ false *is ignorant*

$\mathrm{wp}[\![\mathsf{ann!}\,\phi]\!].\mathsf{false} \Leftrightarrow \mathrm{wp}[\![\mathsf{ann!}\,\phi \mathbin{⨟} \mathsf{ann!}\,\psi]\!].\mathsf{false}$

$\Rrightarrow$

$[\phi]\mathsf{false} \Leftrightarrow [\langle\phi\rangle\psi]\mathsf{false}$

$\Rrightarrow$

$\neg\phi \Leftrightarrow \neg\langle\phi\rangle\psi$

$\Rrightarrow$

$\phi \Leftrightarrow \langle\phi\rangle\psi$

$\Rrightarrow$

$\mathsf{true} \Leftrightarrow [\phi]\psi$

$\Rrightarrow$

$[\phi]\psi$

$\square$

Soundness of Law 5.15 . Using the wp semantics, the first equality is equivalent to $\langle\phi\rangle\psi \Leftrightarrow \langle\phi\rangle(\phi \wedge \psi)$, which obtains for a successful formula ϕ. Indeed,

$\mathsf{true} \Leftrightarrow [\phi]\phi$

$\Rrightarrow$

$\langle\phi\rangle\psi \Leftrightarrow ([\phi]\phi \wedge \langle\phi\rangle\psi)$

$\Rrightarrow$

$\langle\phi\rangle\psi \Leftrightarrow ([\phi]\phi \wedge \phi \wedge \langle\phi\rangle\psi)$

$\Rrightarrow$

$\langle\phi\rangle\psi \Leftrightarrow (\langle\phi\rangle\phi \wedge \langle\phi\rangle\psi)$

$\Rrightarrow$

$\langle\phi\rangle\psi \Leftrightarrow \langle\phi\rangle(\phi \wedge \psi)$

For the two other equalities, we use properties of successful formulas, already observed for example in [11]: if ϕ is successful then $[\phi]\phi$ is valid if and only if $[\phi]\mathrm{C}\phi$ is. $\square$

Soundness of Law 5.11. For a formula φ not containing K_a or KV_a, $\langle\psi\rangle\varphi = \psi \wedge \varphi$. This was observed in the propositional case, for example in [33]. It can also be obtained for φ with quantifiers by induction on φ. $\square$

Chapter 6

Applications

6.1 The Three Wise Men puzzle

Imagine three wise men are tested by their king as follows. The king puts a hat on the head of each wise man. Each wise man can see the hat of the others but not his own. The king announces that hats are either black or white and at least one of them is black. The king then asks the first wise man if he knows his hat's color or not, and the first wise man replies "No". The king went on to ask the second wise man, who also does not know his hat's color. In this situation, the third wise man learns that he has a black hat. In fact everyone learns that, and it is a common knowledge, assuming that all the questions and declarations are public. We note that, in this puzzle the wise men are assumed to be ideal agents, their knowledge obeys the axioms of S5 (Section 1.4), and that they can carry out logical reasoning. Figure 6.1 provides description of the puzzle using the programming language of Chapter 2.

$$\begin{array}{lcl} \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{⨟} & & \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{⨟} \\ \mathsf{ann!}\,\neg \mathrm{KV}_0 p_0 \mathbin{⨟} & = & \mathsf{ann!}\,\neg \mathrm{KV}_0 p_0 \mathbin{⨟} \mathsf{ann!}\,(p_1 \vee p_2) \mathbin{⨟} \\ \mathsf{ann!}\,\neg \mathrm{KV}_1 p_1 & & \mathsf{ann!}\,\neg \mathrm{KV}_1 p_1 \mathbin{⨟} \mathsf{ann!}\,p_2 \end{array}$$

FIGURE 6.1 The Three Wise Men puzzle as a program equation. Boolean variables $(p_i)_{i=0,\dots,2}$ correspond to the hat's color of the wise men $(A_i)_{i=0,\dots,2}$ such that p_i is true when A_i wears a black hat. The programs are inside the scope of $p_i \in vis.p_j \equiv i \neq j$.

The left hand side of 6.1 describes the successive announcements by the king, by A_0 who does not know his hat's color, and by A_1 who also does not know his hat's color. The right hand side of the equation tells us that after the announcement by A_0 it is revealed (to all) that $p_1 \vee p_2$, and after the announcement by A_1 it is revealed (to all) that p_2. Consequently, the third wise man A_2 learns that he wears a black hat (p_2). Now we provide a derivation of this equation using the programming and logical laws from the previous chapters.

Proof of the equation in Figure 6.1.

$$\begin{array}{ll}
& \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_0\, p_0 \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_1 p_1 \\
= & \hfill \textit{Law 5.15, } p_0 \vee p_1 \vee p_2 \textit{ is successful} \\
& \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_0\, p_0 \wedge \mathrm{K}_0(p_0 \vee p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_1 p_1 \\
= & \hfill p \in vis \Rightarrow \mathrm{K}(p \vee q) \leftrightarrow p \vee \mathrm{K}q \ \textit{(Law 3.3)} \\
& \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_0\, p_0 \wedge (\mathrm{K}_0 p_0 \vee (p_1 \vee p_2)) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_1 p_1 \\
= & \hfill \neg\mathrm{KV}_0\, p_0 \wedge \mathrm{K}_0 p_0 \leftrightarrow \mathsf{false} \\
& \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \\
& \mathsf{ann!}\,(\neg\mathrm{KV}_0\, p_0 \wedge (p_1 \vee p_2)) \\
& \mathsf{ann!}\,\neg\mathrm{KV}_1 p_1 \\
= & \hfill \textit{For atomic formula } p\textit{:} \\
& \hfill \mathsf{ann!}\,\phi \wedge p = \mathsf{ann!}\,\phi \mathbin{;} \mathsf{ann!}\,p \\
& \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_0\, p_0 \mathbin{;} \mathsf{ann!}\,(p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_1 p_1 \\
= & \hfill \textit{Law 5.15, } p_1 \vee p_2 \textit{ is successful} \\
& \mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \mathbin{;} \\
& \mathsf{ann!}\,\neg\mathrm{KV}_0\, p_0 \mathbin{;} \mathsf{ann!}\,(p_1 \vee p_2) \\
& \mathsf{ann!}\,(\neg\mathrm{KV}_1 p_1 \wedge \mathrm{K}_1(p_1 \vee p_2))
\end{array}$$

$=$ $\quad p \in \mathit{vis} \Rightarrow \mathrm{K}(p \vee q) \leftrightarrow p \vee \mathrm{K}q$ *(Law 3.3)*

$$\mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \,⨾\, \mathsf{ann!}\,\neg \mathrm{KV}_0\, p_0 \,⨾\, \mathsf{ann!}\,(p_1 \vee p_2) \, \mathsf{ann!}\,(\neg \mathrm{KV}_1 p_1 \wedge (\mathrm{K}_1 p_1 \vee p_2))$$

$=$ $\quad \neg \mathrm{KV}_1 p_1 \wedge \mathrm{K}_1 p \Leftrightarrow \mathsf{false}$

$$\mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \,⨾\, \mathsf{ann!}\,\neg \mathrm{KV}_0\, p_0 \,⨾\, \mathsf{ann!}\,(p_1 \vee p_2) \, \mathsf{ann!}\,(\neg \mathrm{KV}_1 p_1 \wedge p_2))$$

$=$ *For atomic formula p:* $\mathsf{ann!}\,\phi \wedge p = \mathsf{ann!}\,\phi \,⨾\, \mathsf{ann!}\,p$

$$\mathsf{ann!}\,(p_0 \vee p_1 \vee p_2) \,⨾\, \mathsf{ann!}\,\neg \mathrm{KV}_0\, p_0 \,⨾\, \mathsf{ann!}\,(p_1 \vee p_2) \,⨾\, \mathsf{ann!}\,\neg \mathrm{KV}_1 p_1 \,⨾\, \mathsf{ann!}\,p_2$$

It is possible, in this proof, not to use KV and replace instances of $\mathrm{KV}p$ with $\mathrm{K}p \vee \mathrm{K}\neg p$. □

In this puzzle, we could observe that the successive announcements, by the king and by the first two wise men, induce some information flow to the third wise man. The latter could learn the color of his hat even if that was not communicated directly to him. If the three wise men were asked by the king to reply simultaneously to the same question, then all three would not know the color of their hat (if they are asked by the king only once!). This is another difference between simultaneous and successive announcements, similar to what was observed in Example 5.1. But even answering simultaneously to the same question, the wise men can infer information on the color of their hats. This version is called the Muddy Children Puzzle and is our next example.

6.2 The Muddy Children Puzzle

This puzzle was known from [12]. Three children (which generalises to n) get dirty after playing together in the garden. The children want to get clean but they cannot be sure if their forehead is clean, as they cannot see their forehead; but each can see the forehead of any other. Their father comes along, and tells them: "At least one of you has a muddy forehead". After that, the father asks all of them: "Do you know if you have a muddy forehead or not?", and all of them reply at the same time "No". After, the father asking the same question a second time, all children say "No" again, but after a third time, the three of them all know.

Using program algebra and logic (classical, epistemic, and PAL) we will investigate:

- that in this scenario the three children are all muddy,
- how does each of them learn that they are muddy after a precise number of steps?
- what do the children learn after each step of their simultaneous announcements?

6.2.1 Puzzle with three muddy children

The left hand side of the equation in Figure 6.2 describes the announcements by the father and the first two simultaneous announcement by the children. The right hand side shows the implicit flow of information after each of the children's simultaneous announcement. For instance, after their first simultaneous announcement (that each of them does not know whether she is muddy or not), it is revealed that for every two of them, at least one is muddy.

$$\begin{aligned}
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \fatsemi \\
&\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2 \fatsemi \\
&\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2 \\
&= \\
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \fatsemi && (6.2.1) \\
&\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2 \fatsemi && (6.2.2) \\
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0) \fatsemi && (6.2.3) \\
&\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2 \fatsemi && (6.2.4) \\
&\mathsf{ann!}\, p_0 \wedge p_1 \wedge p_2 && (6.2.5)
\end{aligned}$$

FIGURE 6.2 The Three Muddy Children puzzle. The children are denoted by a_0, a_1, a_2 with corresponding modalities $\mathrm{K}_0, \mathrm{K}_1, \mathrm{K}_2$. The programs are in the scope of $p_i \in vis.p_j \equiv i \neq j$, where p_i is a boolean variable that is true when a_i has a muddy forehead.

Proof of the equation in Figure 6.2 . We prove first that (6.2.1) $\fatsemi$ (6.2.2) $=$ (6.2.1) $\fatsemi$ (6.2.2) $\fatsemi$ (6.2.3). Then, we will prove that (6.2.3) $\fatsemi$ (6.2.4) $=$ (6.2.3) $\fatsemi$ (6.2.4) $\fatsemi$ (6.2.5).

$$\begin{aligned}
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \fatsemi \\
&\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2 \\
= \quad & \textit{$p_0 \vee p_1 \vee p_2$ is successful, becomes common knowledge after announcement, Law 5.15} \\
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \fatsemi \\
&\mathsf{ann!}(\neg \mathrm{KV}_0 p_0 \wedge \mathrm{K}_0(p_0 \vee p_1 \vee p_2) \wedge \\
&\qquad \neg \mathrm{KV}_1 p_1 \wedge \mathrm{K}_1(p_1 \vee p_2 \vee p_0) \wedge \\
&\qquad \neg \mathrm{KV}_2 p_2 \wedge \mathrm{K}_2(p_2 \vee p_0 \vee p_1))) \\
= \quad & \textit{p_j and p_k are visible to agent $A_{\{i \neq j,k\}}$} \\
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \fatsemi \\
&\mathsf{ann!}(\neg \mathrm{KV}_0 p_0 \wedge (\mathrm{K}_0 p_0 \vee (p_1 \vee p_2))) \wedge \\
&\qquad \neg \mathrm{KV}_1 p_1 \wedge (\mathrm{K}_1 p_1 \vee (p_2 \vee p_0))) \wedge \\
&\qquad \neg \mathrm{KV}_2 p_2 \wedge (\mathrm{K}_2 p_2 \vee (p_0 \vee p_1)))) \\
= \quad & \neg \mathrm{KV}_0 \wedge \mathrm{K}_0 p_0 = \mathsf{false} \\
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \fatsemi
\end{aligned}$$

$$\begin{aligned}
&\mathsf{ann!}(\neg \mathrm{KV}_0 p_0 \wedge (p_1 \vee p_2))\wedge \\
&\qquad \neg \mathrm{KV}_1 p_1 \wedge (p_2 \vee p_0))\wedge \\
&\qquad \neg \mathrm{KV}_2 p_2 \wedge (p_0 \vee p_1)))
\end{aligned}$$

$=$ *$(p_i \vee p_j)$ is successful, remains true after announcement*

$$\begin{aligned}
&\mathsf{ann!}\, p_0 \vee p_1 \vee p_2 \,⨟ \\
&\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2 \,⨟ \\
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0)
\end{aligned}$$

Now, we continue with the second part of the proof.

$$\begin{aligned}
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0) \,⨟ \\
&\mathsf{ann!}\, (\neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2)
\end{aligned}$$

$=$ *$(p_i \vee p_j)$ is successful, becomes common knowledge after announcement*

$$\begin{aligned}
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0) \,⨟ \\
&\mathsf{ann!}(\neg \mathrm{KV}_0 p_0 \wedge \mathrm{K}_0(p_0 \vee p_1)\wedge \\
&\qquad \neg \mathrm{KV}_1 p_1 \wedge \mathrm{K}_1(p_1 \vee p_2)\wedge \\
&\qquad \neg \mathrm{KV}_2 p_2 \wedge \mathrm{K}_2(p_2 \vee p_0))
\end{aligned}$$

$=$ *p_i is visible to A_j for $j \neq i$*

$$\begin{aligned}
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0) \,⨟ \\
&\mathsf{ann!}(\neg \mathrm{KV}_0 p_0 \wedge (\mathrm{K}_0 p_0 \vee p_1)\wedge \\
&\qquad \neg \mathrm{KV}_1 p_1 \wedge (\mathrm{K}_1 p_1 \vee p_2)\wedge \\
&\qquad \neg \mathrm{KV}_2 p_2 \wedge (\mathrm{K}_2 p_2 \vee p_0))
\end{aligned}$$

$=$ $(\neg \mathrm{KV}_i p_i \wedge \mathrm{K}_i p_i) = \mathsf{false}$

$$\begin{aligned}
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0) \,⨟ \\
&\mathsf{ann!}(\neg \mathrm{KV}_0 p_0 \wedge p_1\wedge \\
&\qquad \neg \mathrm{KV}_1 p_1 \wedge p_2\wedge \\
&\qquad \neg \mathrm{KV}_2 p_2 \wedge p_0)
\end{aligned}$$

$=$ *p_i is a successful formula*

$$\begin{aligned}
&\mathsf{ann!}\, (p_0 \vee p_1) \wedge (p_1 \vee p_2) \wedge (p_2 \vee p_0) \,⨟ \\
&\mathsf{ann!}\, (\neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2) \,⨟ \\
&\mathsf{ann!}\, (p_0 \wedge p_1 \wedge p_2)
\end{aligned}$$

□

This proof generalises to a situation with n children where the father announces that at least m_a of them is muddy.

6.2.2 Puzzle with n children, all muddy

In this case, we assume that

- there are n children
- the father announces: "At least m_a of you is muddy" (Announcement 0)
- all n children says: "No" to the question of their father $n - m_a$ times (Announcement 1 to Announcement $n - m_a$)
- every child learns that she is muddy after Announcement $n - m_a$

We note that we do not assume to know that all n children are muddy. We can deduce that from the fact that every child learns that she is muddy after Announcement $n - m_a$. In fact, the kind of deduction that we make is made from the point of view of a muddy child. When a muddy child makes his reasoning, he does not know *a priori* that all n children are muddy. However, all the other assumptions and the announcements are common knowledge to all n children.

The proof used in this case also serves in the most general case with m muddy children ($m \leq n$).

We use the following notations.

$$N = \{0, 1, \ldots, n-1\} \qquad \text{(Set of indices)}$$
$$\binom{N}{k} = \{\gamma \subseteq N \mid |\gamma| = k\} \qquad \text{(Set of k-elements subsets)}$$

The following equation states what is revealed from the first announcement by the father and the first simultaneous announcement of the children.

$$\begin{aligned}
&\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right) \fatsemi \\
&\mathsf{ann!}\left(\bigwedge_{i\in N} \neg\mathrm{KV}_i p_i\right)
\end{aligned}
\quad = \quad
\begin{aligned}
&\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right) \fatsemi \\
&\mathsf{ann!}\left(\bigwedge_{i\in N} \neg\mathrm{KV}_i p_i\right) \fatsemi \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{1}}\left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right)
\end{aligned}$$

(6.2.6)

The disjunction $(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right))$ reads as "There is at least a combination of m_a children that are all muddy", which can be read as "There are at least m_a muddy children". So, it corresponds to the announcement of the father.

This same announcement can also be put in other words as

for every n children, at least m_a is muddy (Step 0).

And the implicit information flow in 6.2.6 can be read as

for every $n-1$ children, at least m_a is muddy (Step 1).

And in fact, this will generalise at any step j (provided that $j \leq m-a$):

for every $n-j$ children, at least m_a is muddy (Step j).

By induction on the number of steps (simultaneous announcements), we will prove that after $n-m_a$ steps, all children learns that they are muddy. For that, we need to prove the base case 6.2.6.

Proof of Equation 6.2.6.

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right) ⨾$$
$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg\mathrm{KV}_i p_i\right)$$

$=$ *the first formula announced is successful, it is a factual formula*

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right) ⨾$$
$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg\mathrm{KV}_i p_i\right) \wedge \bigwedge_{k\in N}\mathrm{K}_k\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right)$$

$=$ *for each k, separate the conjunctions that contains p_k*

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma} p_j\right)\right) ⨾$$

$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg \mathrm{KV}_i p_i\right)\wedge\bigwedge_{k\in N}\mathrm{K}_k\left(\bigvee_{\gamma\in\binom{N}{m_a}\mid\gamma\ni k}\left(\bigwedge_{j\in\gamma}p_j\right)\vee\bigvee_{\gamma\in\binom{N}{m_a}\mid\gamma\not\ni k}\left(\bigwedge_{j\in\gamma}p_j\right)\right)$$

$=$ *$\wedge p_k$ distributes over $\bigvee$*

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\fatsemi$$
$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg \mathrm{KV}_i p_i\right)\wedge\bigwedge_{k\in N}\mathrm{K}_k\left(\left(p_k\wedge\bigvee_{\gamma\in\binom{N\setminus k}{m_a-1}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\vee\bigvee_{\gamma\in\binom{N\setminus k}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)$$

$=$ *each p_j for $j\in\gamma\in\binom{N\setminus k}{m_a}$ is visible to A_k, Law $\mathrm{K}_k\phi\Leftrightarrow\phi$ applies*

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\fatsemi$$
$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg \mathrm{KV}_i p_i\right)\wedge\bigwedge_{k\in N}\left(\mathrm{K}_k\left(p_k\wedge\bigvee_{\gamma\in\binom{N\setminus k}{m_a-1}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\vee\bigvee_{\gamma\in\binom{N\setminus k}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)$$

$=$ *$\mathrm{K}_k p_k$ is inconsistent with $\neg\mathrm{KV}_k p_k$*

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\fatsemi$$
$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg \mathrm{KV}_i p_i\right)\wedge\bigwedge_{k\in N}\left(\bigvee_{\gamma\in\binom{N\setminus k}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)$$

$=$ *every $k\in N$ corresponds to a $\delta\in\binom{N}{1}$ the disjunction $\bigwedge_{k\in N}\dots$ is a successful formula*

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\fatsemi$$
$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg \mathrm{KV}_i p_i\right)\fatsemi\mathsf{ann!}\bigwedge_{\delta\in\binom{N}{1}}\left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)$$

□

The previous proof is the refinement obtained using the announcement of the father (step 0) and the first simultaneous announcement (step 1). We have a general result any step between 0 and $n - m_a$, namely

$$\begin{array}{l} \mathsf{ann!} \bigwedge_{\delta \in \binom{N}{step}} \left(\bigvee_{\gamma \in \binom{N \setminus \delta}{m_a}} \left(\bigwedge_{j \in \gamma} p_j \right) \right) \fatsemi \\ \mathsf{ann!} \left(\bigwedge_{i \in N} \neg \mathrm{KV}_i p_i \right) \end{array} = \begin{array}{l} \mathsf{ann!} \bigwedge_{\delta \in \binom{N}{step}} \left(\bigvee_{\gamma \in \binom{N \setminus \delta}{m_a}} \left(\bigwedge_{j \in \gamma} p_j \right) \right) \fatsemi \\ \mathsf{ann!} \left(\bigwedge_{i \in N} \neg \mathrm{KV}_i p_i \right) \fatsemi \\ \mathsf{ann!} \bigwedge_{\delta \in \binom{N}{step+1}} \left(\bigvee_{\gamma \in \binom{N \setminus \delta}{m_a}} \left(\bigwedge_{j \in \gamma} p_j \right) \right) \end{array} \tag{6.2.7}$$

The proof of equality (6.2.7) is similar to the previous proof of the refinement (6.2.6) and is found in Appendix A.3. Assuming (6.2.7),

at step 0:

we have $\binom{N}{0} = \{\}$ and

$$\mathsf{ann!} \bigwedge_{\delta \in \binom{N}{0}} \left(\bigvee_{\gamma \in \binom{N \setminus \delta}{m_a}} \left(\bigwedge_{j \in \gamma} p_j \right) \right) = \mathsf{ann!} \left(\bigvee_{\gamma \in \binom{N}{m_a}} \left(\bigwedge_{j \in \gamma} p_j \right) \right) \tag{6.2.8}$$

which corresponds to the announcement of the father: “At least m_a of you have muddy children forehead”.

The general result (6.2.7) used for each step implies the following refinement of the successive announcements.

$$\begin{array}{c} \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{0}} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right)\right) ; \\ n-m_a \text{ times} \left\{ \begin{array}{l} \mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) ; \\ \vdots \\ \mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) \end{array} \right. \end{array} = \begin{array}{l} \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{0}} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right)\right) ; \\ \mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) ; \\ \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{1}} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right)\right) ; \\ \vdots \\ \mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) ; \\ \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{n-m_a}} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right)\right) \end{array} \tag{6.2.9}$$

at step $n - m_a$:

A subset $\delta \in \binom{N}{n-m_a}$ has $n - m_a$ elements. Thus $N \setminus \delta$ has m_a elements, therefore $\binom{N\setminus\delta}{m_a}$ is the singleton $\{N \setminus \delta\}$, and the disjunction vanishes.

$$\begin{aligned} \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{n-m_a}} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right)\right) &= \mathsf{ann!} \bigwedge_{\delta\in\binom{N}{n-m_a}} \left(\bigwedge_{j\in N\setminus\delta} p_j \right) \\ &= \mathsf{ann!} \bigwedge_{\epsilon\in\binom{N}{m_a}} \left(\bigwedge_{j\in\epsilon} p_j \right) \end{aligned}$$

This reveals that for every m_a combination of children, all of them are muddy. Thus for $m_a > 0$, every child learns that she is muddy.

6.2.3 Puzzle with n children, m muddy

For this case, a muddy child learns that they are muddy after step $m - m_a$. Indeed, after step $m - m_a$, we have the following revelation:

$$\mathsf{ann!} \bigwedge_{\delta \in \binom{N}{m-m_a}} \left(\bigvee_{\gamma \in \binom{N \setminus \delta}{m_a}} \left(\bigwedge_{j \in \gamma} p_j \right) \right) \tag{6.2.10}$$

We remark that a subset in $\binom{N\setminus\delta}{m_a}$ has $n - (m - m_a) = (n - m) + m_a$ elements. The revelation (6.2.10) reads that it is revealed that:

> For every $(n - m) + m_a$ children, m_a of them are muddy.

A muddy child A_c sees $n - m$ non-muddy children. If m_a children (which may include A_c) are added to the non-muddy children, all m_a must be muddy. Thus A_c learns that she is muddy.

Thus a muddy child learns that he/she is muddy.

Remark 6.1 The command $\mathsf{ann!} \bigwedge_{i \in N} \neg \mathrm{KV}_i p_i$, reads as it is revealed (truthfully) that no child knows whether she/he is muddy or not. It could be interpreted to be a truthful event as no child steps forward [11]; or by an announcement of the father "none of you knows", assuming that the father has asked privately each child; or that they all announce at the same time: "I don't know" assuming that the children can speak simultaneously and can all hear each other while they are speaking.

6.2.4 Discussion

By describing the scenarios of the puzzle as programs, we preserve their dynamic aspect. The proofs make explicit the information flow between each step of the programs (determined by the sequential composition $\fatsemi$) which is not the case when analysing the puzzle with only epistemic logic. Our proofs make use of only programming laws and logical theorems. We do not reason about the update of the possible worlds (as e.g., in [12]). We could describe faithfully the scenarios of puzzle using the programming language and also encode could the questions as refinements (or equations) between programs.

6.3 The Cocaine Auction Protocol

In this case study, we attempt to formalise an anonymous auction between cocaine dealers from [35].

The scenario

"Several extremely rich and ruthless men are gathered around a table. An auction is about to be held in which one of them will offer his next shipment of cocaine to the highest bidder. The seller describes the merchandise and proposes a starting price. The others then bid increasing amounts until there are no bids for 30 consecutive seconds. At that point the seller declares the auction closed and arranges a secret appointment with the winner to deliver the goods."

This situation represents an extreme case auction in which the trust between the participants is minimum, privacy requirements are of high stake, and cheating could be fatal. The auction requires the following.

- No third party to run the auction, the protocol must be self-enforcing.
- Nobody except the buyer and the seller should know who won the auction.
- The seller should not be able to find out the winner before committing to the sale.

Now we proceed with the implementation of the Cocaine Auction Protocol as proposed in [35].

The protocol

The protocol (Figure 6.3) is organised in timed rounds such that

- the clock is initialised before each round (line 9 or line 12:R:3),
- a round starts just after the seller reveals the new price (line 10 or line 12:R:4),
- a round ends as soon as a buyer says "yes" to the current price or by timeout (R is chosen in line 12 or $t > \Delta t$ in line 11).

In a round, for each interval of time δt

- either a bidder anonymously says "yes" to the current price (line 12:R)
- or nothing happens, the clock ticks (line 12:L).

Everyone would distinguish between the two cases because the "yes" message is public, thus we have modelled the alternative as an explicit choice $\sqcap$. In the first case (right hand side of $\sqcap$), the buyer (from the set A but anonymous, line 12:R:1) broadcasts a "yes" message, which is an encryption of his private key (line 12:R:2). This would allow him to perform a Diffie-Hellman key exchange [9] with the seller, if he eventually becomes the final winner. A new round begins after that.

When a round ends by timeout, the auction terminates and, provided that there was at least a bidder to the initial price (line 14 condition $r > 0$):

- the final winner is the winner of the previous round (line 14:L:1),
- the seller computes a common secret key with the winner (DH-key) with the winner's "yes" message and his secret key (line 14:L:2),
- the seller broadcast an encryption of the date and time of the sale with the DH-key (line 14:L:3).

The winner is the only bidder who can decrypt the appointment message by computing the DH-key (line 15), assuming an encryption method that use the same key for encryption and decryption (i.e., $\mathit{Dec}.(\mathit{Enc}.(\mathit{sale}, \mathit{key})) = \mathit{sale})$.

Discussion

We now discuss one of the possible attacks to this protocol described in [35]: the case where the seller attempts to sell the goods to another buyer other than the final winner. The seller would compute a DH-key with the "yes" message of his preferred buyer, say x_p; then line 14:L:2 would become

$$\mathit{key} := g^{x_p y} \ \mathsf{mod}\ n$$

In such a case, the cheated winner c can expose the treachery by broadcasting publicly his private key ($\mathsf{rev}\{x_c\}$). Everyone can check whether he is the real winner

$$\mathsf{rev}\{x_c = w_f\}$$

$\sqsubseteq ?$

$$\mathsf{rev}\{g^{x_c} \mathsf{\ mod\ } n = g^{w_f} \mathsf{\ mod\ } n\}$$

$\sqsubseteq$ *the "yes" message g^{w_f} mod n was revealed publicly*

$$\mathsf{rev}\{g^{x_c} \mathsf{\ mod\ } n\}$$

$\sqsubseteq$ *g and n are public*

$$\mathsf{rev}\{x_c\}$$

The question mark in the first step indicates that we do not apply a refinement law. What can be achieved is to check whether c has the same "yes" message as the real winner. But it is reasonable to assume that c is the real winner in that case assuming a large n.

Everyone can check if the seller has cheated, by decrypting the seller's appointment message

$$\mathsf{rev}\{Dec.(Enc.(sale, key))\}, g^{x_c y} \mathsf{\ mod\ } n)$$

$\sqsubseteq$ *$Enc.(sale, key)$ was revealed publicly*

$$\mathsf{rev}\{g^{x_c y} \mathsf{\ mod\ } n\}$$

$\sqsubseteq$ *g and n are public*

$$\mathsf{rev}\{x_c\}$$

Variants of the protocol are presented in [35], as well as other possible attacks and the practical interests of such protocols, other than selling drugs.

Whilst it might be difficult to formalise every aspect of the protocol, we could describe it using a formal programming language. We could abstract, for instance, from how the anonymous broadcasts are done. Implementation of these broadcasts based on the Dining Cryptographer's protocol and based on physical devices are discussed in [35]; these are refinements of the program in Figure 6.3.

```
1:  $\mathsf{vis}_{a_0}\ y : \mathsf{D}$ ⨾                                   Secret key of the seller $a_0$
2:  $\mathsf{vis}_A\ g : \mathsf{D}, n : \mathbb{N}, g^y \mathsf{\ mod\ } n$ ⨾          Public keys of the seller $a_0$
3:  $\mathsf{hid}_A\ w : A^N$ ⨾                                 The winner of a round: hidden to all
4:  $\mathsf{hid}_A\ w_f : A$ ⨾                                 The final winner: hidden to all
5:  $\mathsf{vis}_a\ x_a :\in \mathsf{D}^N$ ⨾                            Secret key of agent $a$ at round $i$
6:  $\mathsf{vis}_{a_0}\ sale : \mathsf{D}$ ⨾                           Place and time of the sale: visible to $a_0$
7:  $\mathsf{vis}\ t : \mathbb{N}, r : \mathbb{N}$ ⨾                        The time, the round are visible to all
8:  $\mathsf{vis}_{a_0}\ price : \mathbb{N}^N$ ⨾                        The price for each round: visible to $a_0$
9:  $t, r := 0, 0$ ⨾
10: $\mathsf{rev}\{price[0]\}$ ⨾
11: while $t < \Delta t$ do
12:         L: $t := t + \delta t$ $\sqcap$ R: [ 1: $w[r] :\in A$ ⨾
                                         2: $\mathsf{rev}\{g^{x_{w[r]}} \mathsf{\ mod\ } n\}$
                                         3: $t, r := 0, r + 1$ ⨾
                                         4: $\mathsf{rev}\{price[r]\}$ ] ⨾
13: end while
14: L: [ 1: $w_f := w[r-1]$ ⨾
         2: $key := (g^{x_{w_f}})^y \mathsf{\ mod\ } n$ ⨾
         3: $\mathsf{rev}\{Enc.(sale, key)\}$ ] $\lhd r > 0 \rhd$ R: skip ⨾
15: $key := (g^y)^x_{w_f} \mathsf{\ mod\ } n$
```

FIGURE 6.3: The Cocaine Auction Protocol

Chapter 7

Conclusion

In this work, we have proposed a programming language for refinement calculus in which the underlying predicate logic allows us to express knowledge. The programming language extends the one used in Morgan's Shadow refinement calculus [28] to include, for instance an announcement command ann! and to allow higher order knowledge formulas. For example, the simultaneous answers of the children to their father in the Muddy Children Puzzle was expressed by

$$\mathsf{ann!}\, \neg \mathrm{KV}_0 p_0 \wedge \neg \mathrm{KV}_1 p_1 \wedge \neg \mathrm{KV}_2 p_2$$

We used a first-order epistemic logic that is based on Fitting's First Order Intensional Logic (FOIL) [13, 15]. Our addition was to define explicitly a visibility type for intensions (or concepts) by specifying the set A of agents that see a variable v, in which case we have $v \in \mathit{vis}.A$. We defined a concrete Kripke model using *vis* and a set of intensions (or concepts). The latter determines the possible states and the former determines the accessibility relations. The type of model used here is a multi-dimensional frame in the sense of [22]. The model for the logic was also used for the program denotational semantics. This could be done only by storing all the past values of the program variables, not forgetting local variables. By extending the logic to Public Announcement Logic (PAL, [33, 18, 11]) we could also define a weakest precondition semantics for the programs.

In addition to structural laws, which also apply for classical programs, the laws governing confidentiality/knowledge sensitive programs take into account the information flow to the agents. An agent gets information from how much it knows from the program code executed (limited by atomicity), from the program variables visible to it, and from higher-order information. The laws using a public announcement derive mostly from PAL.

For reasoning about information flow we established a program calculus with refinement. We do not propose a new dynamic epistemic logic but propose that the Public Announcement Logic can be used in conjunction with program algebra to reason about knowledge dynamics. In our framework, epistemic actions can be written in a programming language; reasoning about them is achieved using programming laws and the theorems of PAL. We explained the Wise Men Puzzle and the Muddy Children [12] using this approach. Reasoning with laws saves us from explaining model updates but still gives an operational understanding of the puzzle. What the children can learn at each step of the puzzle is expressed in a logical formula.

For applications in protocol analysis, we described the Cocaine Auction Protocol [35] using the language and techniques of refinement. Reasoning about this anonymous auction protocol demonstrates how can we use the framework to reason about

- confidentiality of data,
- private or anonymous communication,
- hidden computation.

A completion of this framework would introduce iterative and recursive programs, procedure call and other commands used in the refinement calculus. To be investigated are also the decidability of the intensional logic given in Chapter 3, as well as the axiomatization of the first-order Public Announcement Logic. It is also possible to provide a complete set of programming laws, as in [21] for classical programs, and in [34] for a subset of the programming language used here.

This work could constitute a basis for quantitative analysis of information flow in programs. The proposed denotational model is close to Aumann's information models (see [1]). Another development would analyse the information flow in probabilistic systems; in the direction of [31].

Appendix A

Appendix

A.1 First-order Intensional Logic

A.1.1 FOIL with quantification on objects

Definition A.1 (Truth in a Model) Let a model $\mathfrak{M}$ and an assignment μ of free variables

1. $\mathfrak{M}, \mathsf{s}, \mu \vDash R(t_1, \ldots, t_n)$ iff $(\mu(t_1), \ldots, \mu(t_n)) \in \mathfrak{M}[\![R]\!].H$
2. $\mathfrak{M}, \mathsf{s}, \mu \vDash K_a\phi$ iff for every $\mathsf{t} \in \mathsf{S}$, if $\mathsf{s} \sim_a \mathsf{t}$ then $\mathfrak{M}, \mathsf{t}, \mu \vDash \phi$
3. $\mathfrak{M}, \mathsf{s}, \mu \vDash \forall x \bullet \phi$ iff we have $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for every $\nu = \mu \oplus \{x \mapsto a\}$ where $a \in \mathbf{D}_I$
4. $\mathfrak{M}, \mathsf{s}, \mu \vDash \exists x \bullet \phi$ iff we have $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for some $\nu = \mu \oplus \{x \mapsto a\}$ where $a \in \mathbf{D}_I$
5. $\mathfrak{M}, \mathsf{s}, \mu \vDash \forall X \bullet \phi$ iff we have $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for every $\nu = \mu \oplus \{X \mapsto a\}$ where $a \in \mathbf{D}_O$
6. $\mathfrak{M}, \mathsf{s}, \mu \vDash \exists X \bullet \phi$ iff we have $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for some $\nu = \mu \oplus \{X \mapsto a\}$ where $a \in \mathbf{D}_O$
7. $\mathfrak{M}, \mathsf{s}, \mu \vDash (\lambda X \bullet \phi)(t)$ iff we have $\mathfrak{M}, \mathsf{s}, \nu \vDash \phi$ for $\nu = \mu \oplus \{x \mapsto \mathfrak{M}[\![t]\!].\mathsf{s}\}$

A.1.2 Theorems and axioms of FOIL

Proposition A.1 *The following are Axioms of FOIL without quantifiers (not S5 FOIL). Capital letters X, Y designate objects, t designate concept terms.*

1. $(\lambda X \bullet \phi \Rightarrow \psi).t \Rightarrow ((\lambda X \bullet \phi).t \Rightarrow (\lambda X \bullet \psi).t)$
2. *If X is not free in ϕ then* $(\lambda X \bullet \phi).t \Rightarrow \phi$
3. $(\lambda X \bullet \phi.X).t \Rightarrow (\lambda X \bullet \phi.Y).t$
4. $D.t \Rightarrow ((\lambda X \bullet \phi).t \vee (\lambda X \bullet \neg\phi).t)$
5. $X = X$
6. $X = Y \Rightarrow (P.X \equiv P.Y)$
7. $(X = Y) \Rightarrow \mathrm{K}_a(X = Y)$
8. $\neg(X = Y) \Rightarrow \mathrm{K}_a\neg(X = Y)$
9. $D.t \Rightarrow (\lambda X, Y \bullet X = Y).(t, t)$

Proposition A.2 *The following are FOIL theorems from Fitting*

1. $(D.t \wedge \phi) \Rightarrow (\lambda X \bullet \neg\phi).t$ *provided that X is not free in ϕ*
2. $(\lambda X \bullet \neg\phi).t \Rightarrow D.t$
3. $(\lambda X \bullet \neg\phi).t \Leftrightarrow D.t \wedge \phi$ *provided that X is not free in ϕ*
4. $(\lambda X \bullet \neg\phi \wedge \psi).t \Leftrightarrow (\lambda X \bullet \neg\phi).t \wedge (\lambda X \bullet \neg\phi).t$
5. $(\lambda X \bullet \neg\phi).t \Rightarrow \neg(\lambda X \bullet \neg\phi).t$
6. $D.t \Rightarrow ((\lambda X \bullet \neg\phi).t \Leftrightarrow \neg(\lambda X \bullet \neg\phi).t)$
7. $D.t \Rightarrow ((\lambda X \bullet \neg\phi \Rightarrow \psi).t) \Leftrightarrow ((\lambda X \bullet \neg\phi).t \Rightarrow (\lambda X \bullet \neg\psi).t)$
8. $(\lambda X \bullet \phi \vee \psi).t \Leftrightarrow (\lambda X \bullet \neg\phi).t \vee (\lambda X \bullet \neg\phi).t$
9. $((\lambda X \bullet X = Y).t \wedge (\lambda Y \bullet Y = Z).t) \Rightarrow (X = Z)$

Proof of Law 3.3.11. Given $\mathfrak{M}, \mathsf{s}, \mu$, if a term t that designates at H for μ

$\langle p \rangle \mathrm{KV}_a t$

$\Leftrightarrow$ *Law* KV

$\langle p \rangle\, \exists\, y \bullet y \in \mathit{vis}.a \wedge \mathrm{K}_a(t \doteq y)$

Law 3.3.4 and Law 3.3.7

$\Leftrightarrow$

as y is not free in p

$\exists\, y \bullet \langle p\rangle y \in \mathit{vis}.a \wedge \langle p\rangle \mathrm{K}_a(t \doteq y)$

Law 3.3.4 and Law 3.3.7

$\Leftrightarrow$

as y is not free in p

$\exists\, y \bullet \langle p\rangle y \in \mathit{vis}.a \wedge \langle p\rangle \mathrm{K}_a(t \doteq y)$

$\Leftrightarrow$ *Law 3.3.2*

$\exists\, y \bullet p \wedge y \in \mathit{vis}.a \wedge \langle p\rangle \mathrm{K}_a(t \doteq y)$

$\Leftrightarrow$ *Law 3.3.10*

$\exists\, y \bullet p \wedge y \in \mathit{vis}.a \wedge (p \wedge \mathrm{K}_a(p \rightarrow \langle p\rangle t \doteq y))$

$\Leftrightarrow$ *Law 3.3.2*

$\exists\, y \bullet p \wedge y \in \mathit{vis}.a \wedge (p \wedge \mathrm{K}_a(p \rightarrow (p \wedge t \doteq y)))$

p implies q

$\Leftrightarrow$

iff p implies (p and q)

$p \wedge \exists\, y \in \mathit{vis}.a \bullet \mathrm{K}_a(p \rightarrow (t \doteq y))$

If a term t does not designate at H for μ the equivalence still holds. $\square$

Law 3.2: $\mathrm{K}_a(u \doteq v) \wedge \mathrm{K}_a(v \doteq w) \Rightarrow \mathrm{K}_a(u \doteq w)$.

$\mathrm{K}_a(\lambda U, V \bullet U = V).(u, v) \wedge \mathrm{K}_a(\lambda V, W \bullet V = W).(v, w)$

$\Rightarrow$

$\mathrm{K}_a[(\lambda U, V \bullet U = V).(u, v) \wedge (\lambda V, W \bullet V = W).(v, w)]$

$\Rightarrow$ *Axiom 4.1.4 Fitting*

$\mathrm{K}_a[(\lambda U, W \bullet (\lambda V \bullet U = V \wedge \lambda V \bullet V = W).y).(x, z)]$

$\Rightarrow$ *Axiom 4.1.9 Fitting*

$\mathrm{K}_a[(\lambda U, W \bullet U = W).(x, z)]$

$\Rightarrow$

$\mathrm{K}_a(x \doteq z)]$

$\square$

A.2 Soundness of PAL axioms in Chapter 3

In the following we prove the soundness of Laws 3.3.2, 3.3.3, 3.3.4, 3.3.10, 3.3.8, 3.3.9. Laws 3.3.5 and 3.3.7 can be derived from the first set of laws. Derivation of Law 3.3.11 is presented.

Proof of Law 3.3.2. Given $\mathfrak{M}, \mathsf{s}, \mu$, if $t_0, \ldots, t_{N-1}$ designates at s for μ then

$\quad \mathfrak{M}, \mathsf{s}, \mu \models \langle p \rangle R(t_0, \ldots, t_{N-1})$

$\equiv$ *Definition 3.17*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } \mathfrak{M}_{|p}, \mathsf{s}, \mu \models R(t_0, \ldots, t_{N-1})$

$\equiv$ *Truth of an atomic formula*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } (\mu(t_0), \ldots, \mu(t_{N-1})) \in \mathfrak{M}_{|p}[\![R]\!].\mathsf{s}$

$\equiv$ *p and p iff p*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } (\mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } (\mu(t_0), \ldots, \mu(t_{N-1})) \in \mathfrak{M}_{|p}[\![R]\!].\mathsf{s})$

$\equiv$ *if $\mathfrak{M}, \mathsf{s}, \mu \models p$ then $\mathsf{s} \in \operatorname{dom} \mathsf{S}_{|p}$ and $\mathfrak{M}_{|p}[\![\,.\,]\!] = \mathsf{S}_{|p} \lhd \mathfrak{M}[\![\,.\,]\!]$*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } (\mu(t_0), \ldots, \mu(t_{N-1})) \in \mathfrak{M}[\![R]\!].\mathsf{s}$

$\equiv$ *Truth of an atomic formula and $\wedge$*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } R(t_0, \ldots, t_{N-1})$

If some t_i does not designate, the equivalence remains. □

Proof of Law 3.3.3.

$\quad \mathfrak{M}, \mathsf{s}, \mu \models \langle p \rangle \neg \phi$

$\equiv$ *Definition 3.17*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } \mathfrak{M}_{|p}, \mathsf{s}, \mu \models \neg \phi$

$\equiv$ *Truth of negation, p and q iff p and $(\neg p$ or $q)$*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } (\mathfrak{M}, \mathsf{s}, \mu \not\models p \vee \mathfrak{M} \mid p, \mathsf{s}, \mu \not\models \phi)$

$\equiv$ *De Morgan's law*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } \neg(\mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } \mathfrak{M} \mid p, \mathsf{s}, \mu \models \phi)$

$\equiv$ *Definition 3.17*

$\quad \mathfrak{M}, \mathsf{s}, \mu \models p \text{ and } \mathfrak{M}, \mathsf{s}, \mu \not\models \langle p \rangle \phi$

$\equiv$ *Truth of* $\wedge$

$\mathfrak{M}, \mathsf{s}, \mu \models p \wedge \neg\langle p\rangle\phi$

$\square$

Proof of Law 3.3.4.

$\mathfrak{M}, \mathsf{s}, \mu \models \langle p\rangle\phi_0 \wedge \phi_1$

$\equiv$ *Definition 3.17*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \phi_0 \wedge \phi_1$

$\equiv$ *Truth of* $\wedge$

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \phi_0$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \phi_1$

$\equiv$ *p iff p and p*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \phi_0$ and $\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \phi_1$

$\equiv$ *Definition 3.17, truth of* $\wedge$

$\mathfrak{M}, \mathsf{s}, \mu \models \langle p\rangle\phi_0 \wedge \langle p\rangle\phi_1$

$\square$

Proof of Law 3.3.10.

$\mathfrak{M}, \mathsf{s}, \mu \models \langle p\rangle \mathrm{K}\phi$

$\equiv$ *Definition 3.17*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \mathrm{K}\phi$

$\equiv$ *Truth of* K

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and for any $\mathsf{t} \sim_{a|p} \mathsf{s}$ we have $\mathfrak{M}_{|p}, \mathsf{t}, \mu \models \phi$

$\equiv$ *Definition of* $\sim_{a|p}$

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and for any $\mathsf{t} \sim_a \mathsf{s}$ and $\mathsf{t} \in \mathsf{S}_{|p}$ we have $\mathfrak{M}_{|p}, \mathsf{t}, \mu \models \phi$

$\equiv$ *(p and q) implies r iff p implies (q implies r)*

$\mathfrak{M}, \mathsf{t}, \mu \models p$ and for any $\mathsf{t} \sim_a \mathsf{s}$, if $\mathsf{t} \in \mathsf{S}_{|p}$ then $\mathfrak{M}_{|p}, \mathsf{t}, \mu \models \phi$

$\equiv$ *Definition of* $\mathsf{S}_{|p}$

$\mathfrak{M}, \mathsf{t}, \mu \models p$ and for any $\mathsf{t} \sim_a \mathsf{s}$, if $\mathfrak{M}, \mathsf{t}, \mu \models p$ then $\mathfrak{M}_{|p}, \mathsf{t}, \mu \models \phi$

$\equiv$ *(p and q) implies r iff p implies (q implies r)*

$\mathfrak{M}, \mathsf{t}, \mu \models p$ and for any $\mathsf{t} \sim_a \mathsf{s}$, if $\mathfrak{M}, \mathsf{t}, \mu \models p$ then $\mathfrak{M}, \mathsf{t}, \mu \models p$ implies $\mathfrak{M}_{|p}, \mathsf{t}, \mu \models \phi$

$\equiv$ *Definition 3.17*

$\mathfrak{M}, \mathsf{t}, \mu \models p$ and for any $\mathsf{t} \sim_a \mathsf{s}$, if $\mathfrak{M}, \mathsf{t}, \mu \models p$ then $\mathfrak{M}, \mathsf{t}, \mu \models \langle p \rangle \phi$

$\equiv$ *Truth of $\Rightarrow$ and $\wedge$*

$\mathfrak{M}, \mathsf{s}, \mu \models p \wedge \mathrm{K}(p \to \langle p \rangle \phi)$

□

Proof of Law 3.3.8.

$\mathfrak{M}, \mathsf{s}, \mu \models \langle p \rangle \forall x \phi$

$\equiv$ *Definition 3.17*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models \forall x \phi$

$\equiv$ *Truth of universal quantification*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \nu \models \phi$ for any $\nu = \mu \oplus \{x \mapsto c\}$ where $c \in \mathsf{D}_c$

$\equiv$ *x is not free in p*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \nu \models \phi$ for any $\nu = \mu \oplus \{x \mapsto c\}$ where $c \in \mathsf{D}_c$

$\equiv$ *Definition 3.17*

$\mathfrak{M}, \mathsf{s}, \mu \models \langle p \rangle \phi$ for any $\nu = \mu \oplus \{x \mapsto c\}$ where $c \in \mathsf{D}_c$

$\equiv$ *Truth of universal quantification*

$\mathfrak{M}, \mathsf{s}, \mu \models \forall x \langle p \rangle \phi$

□

Proof of Law 3.3.9. Given $\mathfrak{M}, \mathsf{s}, \mu$, if a term t that designates at H for μ

$\mathfrak{M}, \mathsf{s}, \mu \models \langle p \rangle (\lambda X \bullet \phi).t$

$\equiv$ *Definition 3.17*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \mu \models (\lambda X \bullet \phi).t$

$\equiv$ *Truth of predicate abstraction*

$\mathfrak{M}, \mathsf{s}, \mu \models p$ and $\mathfrak{M}_{|p}, \mathsf{s}, \nu \models \phi$ where $\nu = \mu \oplus \{X \mapsto (\mathfrak{M}, \mu)[\![t]\!]\}$

$\equiv$ *X is not free in p*

$$\mathfrak{M}, \mathsf{s}, \nu \models p \text{ and } \mathfrak{M}_{|p}, \mathsf{s}, \nu \models \phi \text{ where } \nu = \mu \oplus \{X \mapsto (\mathfrak{M}, \mu)[\![t]\!]\}$$

$\equiv$ *Definition 3.17*

$$\mathfrak{M}, \mathsf{s}, \nu \models \langle p \rangle \phi \text{ where } \nu = \mu \oplus \{X \mapsto (\mathfrak{M}, \mu)[\![t]\!]\}$$

$\equiv$ *Truth of predicate abstraction*

$$\mathfrak{M}, \mathsf{s}, \nu \models (\lambda\bullet\, X \,\langle p \rangle \phi).t$$

If a term t does not designate at H for μ the equivalence still holds. $\square$

A.3 Proofs from Chapter 6

Proof of 6.2.9.

$$\mathsf{ann!} \bigwedge_{\delta\in\binom{N}{step}} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right) \fatsemi$$
$$\mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right)$$

$=$

$$\mathsf{ann!} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right) \fatsemi$$
$$\mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) \wedge \bigwedge_{\delta\in\binom{N}{step}} \bigwedge_{k\in N\setminus\delta} \mathrm{K}_k \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right)$$

$=$

$$\mathsf{ann!} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right) \fatsemi$$
$$\mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) \wedge \bigwedge_{\delta\in\binom{N}{step}} \bigwedge_{k\in N\setminus\delta} \mathrm{K}_k \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a} \mid \gamma\ni k} \left(\bigwedge_{j\in\gamma} p_j \right) \vee \bigvee_{\gamma\in\binom{N\setminus\delta}{m_a} \mid \gamma\not\ni k} \left(\bigwedge_{j\in\gamma} p_j \right) \right)$$

$=$

$$\mathsf{ann!} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right) \fatsemi$$
$$\mathsf{ann!} \left(\bigwedge_{i\in N} \neg \mathrm{KV}_i p_i \right) \wedge \bigwedge_{\delta\in\binom{N}{step}} \bigwedge_{k\in N\setminus\delta} \mathrm{K}_k \left(\left(p_k \wedge \bigvee_{\gamma\in\binom{N\setminus\{\delta\cup\{k\}\}}{m_a-1}} \left(\bigwedge_{j\in\gamma} p_j \right) \right) \vee \bigvee_{\gamma\in\binom{N\setminus\{\delta\cup\{k\}\}}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right)$$

$=$

$$\mathsf{ann!} \left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}} \left(\bigwedge_{j\in\gamma} p_j \right) \right) \fatsemi$$

$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg\mathrm{KV}_i p_i\right)\wedge\bigwedge_{\delta\in\binom{N}{step}}\bigwedge_{k\in N\setminus\delta}\left(\mathrm{K}_k\left(p_k\wedge\bigvee_{\gamma\in\binom{N\setminus\{\delta\cup\{k\}\}}{m_a-1}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\right)\vee\bigvee_{\gamma\in\binom{N\setminus\{\delta\cup\{k\}\}}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\Bigg)$$

$$=$$

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\fatsemi$$

$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg\mathrm{KV}_i p_i\right)\wedge\bigwedge_{\delta\in\binom{N}{step}}\bigwedge_{k\in N\setminus\delta}\left(\bigvee_{\gamma\in\binom{N\setminus\{\delta\cup\{k\}\}}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)$$

$$=$$

$$\mathsf{ann!}\left(\bigvee_{\gamma\in\binom{N\setminus\delta}{m_a}}\left(\bigwedge_{j\in\gamma}p_j\right)\right)\fatsemi$$

$$\mathsf{ann!}\left(\bigwedge_{i\in N}\neg\mathrm{KV}_i p_i\right)\fatsemi\ \mathsf{ann!}\bigwedge\nolimits_{\delta\in\binom{N\setminus\delta}{step+1}}\left(\bigvee\nolimits_{\gamma\in\binom{N\setminus\delta}{m_a}}\left(\bigwedge\nolimits_{j\in\gamma}p_j\right)\right)$$

$\square$

Bibliography

[1] R.J. Aumann. Interactive epistemology I: Knowledge. *International Journal of Game Theory*, 28:263–300, 1999.

[2] R.-J. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer-Verlag, Graduate texts in computer science, 1998.

[3] A. Baltag, L.S. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. Technical report, SEN-R9922, CWI, Amsterdam, 1999.

[4] J. van Benthem. One is a lonely number: on the logic of communication. Technical report, ILLC, University of Amsterdam, 2002.

[5] J. van Benthem. *Modal Logic for Open Minds*. CSLI lecture notes. Center for the Study of Language and Information, 2010.

[6] J. van Benthem, J.D. Gerbrandy, T. Hoshi, and E. Pacuit. Merging frameworks for interaction. *Journal of Philosophical logic*, 2009.

[7] P. Blackburn and J. van Benthem. Modal logic: a semantic perspective. In *Handbook of Modal Logic*. Elsevier, 2007.

[8] P. Blackburn, M. De Rijke, and Y. Venema. *Modal Logic: Graph. Darst*, volume 53. Cambridge University Press, 2002.

[9] W. Diffie and M. E. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[10] E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[11] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*. Synthese Library. Springer, 2007.

[12] R. Fagin, J.Y. Halpern, Y.O. Moses, and M.Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[13] M. Fitting. First-order intensional logic. *Annals of pure and applied logic*, 127(1):171–193, 2004.

[14] M. Fitting. Foil axiomatized. *Studia Logica*, 84(1):1–22, 2006.

[15] M. Fitting. Intensional logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2015 edition, 2015.

[16] M. Fitting and R. Mendelsohn. *First-Order Modal Logic*. Kluwer, 1998.

[17] J. Gerbrandy. *Bisimulations on Planet Kripke*. PhD thesis, University of Amsterdam, 1999.

[18] J.D. Gerbrandy and W. Groeneveld. Reasoning about information change. *Journal of Logic, Language, and Information*, 6:147–169, 1997.

[19] J. Hintikka. *Knowledge and Belief*. Cornell University Press, 1962.

[20] C.A.R. Hoare, I.J. Hayes, He Jifeng, C.C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sorensen, J.M. Spivey, and B.A. Sufrin. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987.

[21] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice Hall International Series in Computer Science. Prentice Hall, 1998.

[22] M. Marx and Y. Venema. *Multi-Dimensional Modal Logic*. Kluwer Academic Publishers, 1997.

[23] A.K. Mciver. The Secret Art of Computer Programming. In *Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing*, ICTAC '09, pages 61–78. Springer-Verlag, 2009.

[24] A.K. McIver, L.A. Meinicke, and C.C. Morgan. Security, probability and nearly fair coins in the Cryptographers' Café. In *Proceedings of the 2nd World Congress on Formal Methods*, Lecture Notes in Computer Science, pages 41–71. Springer-Verlag, 2009.

[25] A.K. McIver and C.C. Morgan. Sums and Lovers: Case studies in security, compositionality and refinement. In *Proceedings of the 2nd World Congress on Formal Methods*, FM '09, pages 289–304. Springer-Verlag, 2009.

[26] A.K. McIver and C.C. Morgan. Compositional refinement in agent-based security protocols. *Formal Aspect of Computing*, 23(6):711–737, 2011.

[27] C.C. Morgan. *Programming from Specifications*. Prentice Hall International Series in Computer Science. Prentice Hall, 2 edition, 1994.

[28] C.C. Morgan. The Shadow Knows: Refinement of ignorance in sequential programs. In *Mathematics of Program Construction*, Lecture Notes in Computer Science, pages 359–378. Springer, 2006.

[29] C.C. Morgan. The Shadow Knows: Refinement and security in sequential programs. *Science of Computer Programming*, 74:629–653, 2009.

[30] C.C. Morgan. Compositional noninterference from first principles. *Formal Aspect of Computing*, 24(1):3–26, 2012.

[31] C.C. Morgan and A.K. McIver. *Abstraction, Refinement And Proof For Probabilistic Systems (Monographs in Computer Science)*. Springer-Verlag, 2004.

[32] R. Parikh and R. Ramanujam. A knowledge based semantics of messages. *Journal of Logic, Language and Information*, 12:453–467, 2003.

[33] J.A. Plaza. Logics of public communications. *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems*, 1989.

[34] S. F. Rajaona. An Algebraic Framework for Reasoning about Security. Master's thesis, University of Stellenbosch, 2013.

[35] F. Stajano and R. Anderson. The cocaine auction protocol: On the power of anonymous broadcast. In *Information Hiding*, pages 434–447. Springer, 2000.